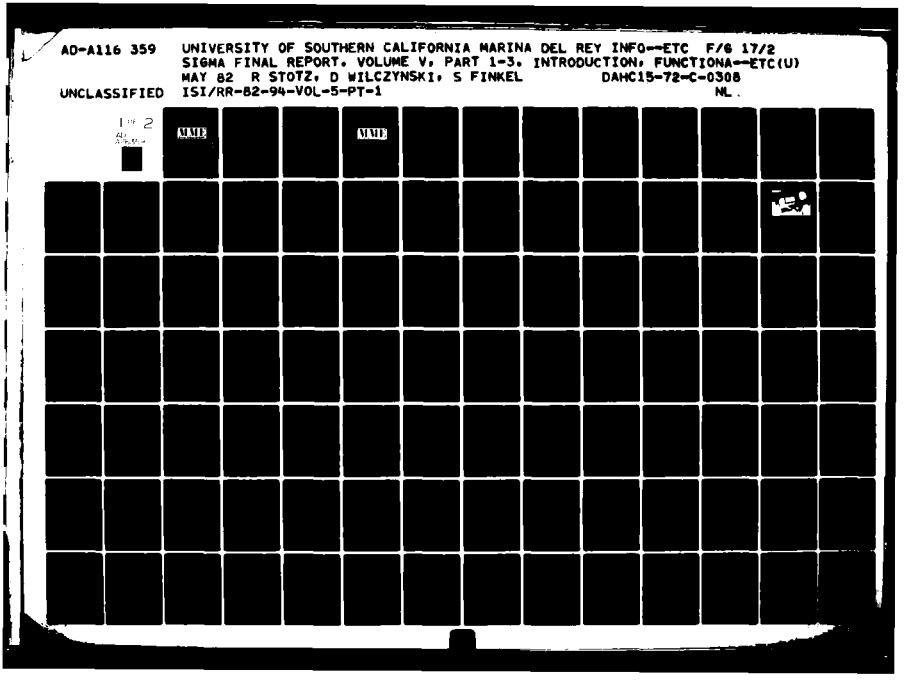
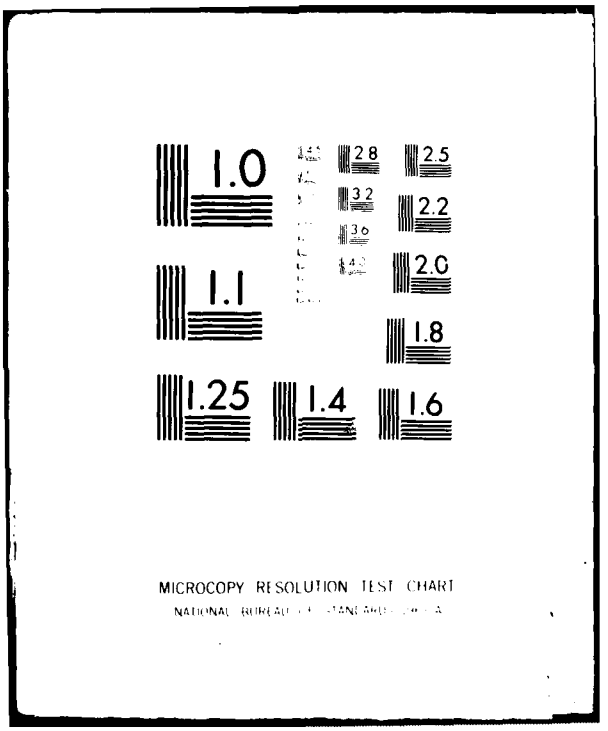
AD-A116 359 UNIVERSITY OF SOUTHERN CALIFORNIA MARINA DEL REY INFO--ETC F/G 17/2
SIGMA FINAL REPORT. VOLUME V. PART 1-3. INTRODUCTION. FUNCTIONA--ETC(U)
MAY 82 R STOTZ, D WILCZYNSKI, S FINKEL DAHC15-72-C-0308
UNCLASSIFIED ISI/RR-82-94-VOL-5-PT-1 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

MME Final Report
Volume V, Parts 1, 2, and 3
ISI/RR-82-94

MME

MILITARY MESSAGE EXPERIMENT

SIGMA Final Report:

Introduction, Functional Description, and Evaluation

Robert Stotz
David Wilczynski
Steven Finkel
Robert Lingard
Donald Oestreicher
Leroy Richardson
Ronald Tugender

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ISI/RR-82-94	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* SIGMA Final Report: Introduction, Functional Description, and Evaluation		5. TYPE OF REPORT & PERIOD COVERED Final Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)* Robert Stotz, David Wilczynski, Steven Finkel, Robert Lingard, Donald Oestreicher, Leroy Richardson, Ronald Tugender		8. CONTRACT OR GRANT NUMBER*(s)* DAHC 15 72 C 0308
9. PERFORMING ORGANIZATION NAME AND ADDRESS USC/Information Sciences Institute 4676 Admiralty Way Marina del Rey, CA 90291		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE May 1982 13. NUMBER OF PAGES 115
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)* ----------		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

This document is approved for public release and sale; distribution is unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

----

18. SUPPLEMENTARY NOTES

----

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

automated message handling, daemons, database organization, DEC PDP-10, editing, file system, HP/MME terminal, Hewlett-Packard 2649 terminal, interactive message processing, interactive message service, interactive terminal, message processing, military communications, Military

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The first part of this report introduces SIGMA, the automated message-handling system used in the Military Message Experiment, developed at the Information Sciences Institute. This introduction is divided into two parts. The first, from 1968 to 1975, covers the period from the recognition of the need for improved communications at Camp Smith, Oahu, to the actual signing of a Memorandum of Agreement to conduct the MME. The second part covers ISI's involvement in the planning and the actual conducting of the MME, roughly from 1973 to 1979.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

19. KEY WORDS (continued)

automated message handling, interactive message service, military communications, Military Message Experiment, nonprofessional computer users, reliability, SIGMA message service, TENEX, terminal-based message service, utility

20. ABSTRACT (continued)

The second part of the *SIGMA Final Report* describes the functionality of SIGMA as a user views it. This part introduces the reader to the system in roughly the sequence that a new user is exposed to it. it starts with a discussion of the terminal, followed by the log_on procedure, then proceeds to the various objects the user deals with in SIGMA and the operations he may perform on them.

The developers of SIGMA learned a great deal during the MME about what the proper functions of an automated message-handling system should be, but these lessons were only part of the developers' education. The experimental results were affected more by several higher level issues than by the details of the message service operation. This part of the *SIGMA Final Report* is divided into the following major sections: high-level issues, functional and design considerations for a message service, and lessons on development and operational environment for the experiment. Parts one, two, and four of the *SIGMA Final Report* are factual; this part, on the other hand, primarily contains opinions of the authors (all members of the ISI team that developed SIGMA), formed from their review of data abstracted from the user interviews, discussions with users, and other peripheral observations.

MME Final Report
Volume V, Parts 1, 2, and 3
ISI/RR-82-94

MME

MILITARY MESSAGE EXPERIMENT

SIGMA Final Report:

Introduction, Functional Description, and Evaluation

Robert Stotz
David Wilczynski
Steven Finkel
Robert Lingard
Donald Oestreicher
Leroy Richardson
Ronald Tugender

This research is supported by the Defense Advanced Research Projects Agency under Contract No. DAHC15 72 C 0308. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. Government, or any person or agency connected with them.

CONTENTS

ACKNOWLEDGMENTS

The list of people who contributed to the Military Message Experiment and thereby to this report is too long to publish. A few people, however, deserve special recognition for their help in producing this document. We especially appreciated the efforts of Dr. Nancy Bryan for her editorial contributions, Jim Melancon for his assistance in the publication process, and Mel Pirtle for moral and intellectual support in traversing the maze of the formatting program and the Penguin printer.

PREFACE

This document comprises Parts One, Two, and Three of Volume V of the MME Final Report. The volumes of the MME Final Report and their topics are:

Volume I	Executive Summary
II	Final Report
III	User View
IV	Message System Utility
V	SIGMA Final Report
VI	Data Analysis and Discussion
VII	Training

The opinions expressed in this report are those of the authors and do not necessarily represent those of USC/Information Sciences Institute or the MME project sponsors. Readers interested in obtaining the remaining volumes of the MME Final Report should contact:

Naval Research Laboratory
Washington, DC 20375
Attn: Code 7590

PART ONE:

INTRODUCTION TO SIGMA

1.1 THE DEVELOPMENT OF SIGMA

The following discussion is the history of the Military Message Experiment as seen through the eyes of the system developers at the Information Sciences Institute (ISI). Section 1.2 describes other documents that will provide a wider perspective of the experiment.

This history is divided into two parts. The first, from 1968 to 1975, covers the period from the recognition of the need for improved communications on Oahu to the actual signing of a Memorandum of Agreement to conduct the MME. The second part covers ISI's involvement in the planning and the actual conducting of the experiment, roughly from 1973 to 1979.

1.1.1 Background

The MME had its roots in the Pueblo incident of early 1968. As a result of Congress' special investigation into the causes of that incident, the Secretary of Defense was directed to improve military communications on the island of Oahu, which is the central focus of communications for all forces in the Pacific. The memo from the Secretary of Defense identifying the need for a program to consolidate communications on Oahu signals the beginning of the COTCO (Consolidation of Telecommunications on Oahu) program. The Assistant to the Secretary of Defense for Telecommunications (ASD Telecommunications), the Joint Chiefs of Staff, and CINCPAC (Commander in Chief, Pacific) exchanged a series of memoranda and communiques to establish plans for development of COTCO; the Navy was given the task of assembling a specific program.

In late 1972-early 1973 (four years after the original memorandum), the director of DARPA's Information Processing Techniques Office was invited to comment on the COTCO requirement. He felt that an interactive message-handling system like the functioning, successful ARPANET message service might be useful on Oahu. ISI was asked to study the problem, and a team of four spent two weeks on Oahu, primarily at CINCPAC Headquarters, studying the state of communications. Their report is *A Plan for Consolidation and Automation of Military Telecommunications on Oahu* [8].

The report proposes radical solutions to the communications problem: distribute around the island some 2000 terminals connected (via links similar to those used in the ARPANET) to five computers dedicated to message processing, which would themselves receive network communications from both island and mainland. Next, give the communications staff access to the terminals and (it is implied) instruct the staff in the use of an already available computerized service for producing and receiving messages. The message processing task itself was not studied in detail. The ARPANET message service was assumed to be sufficient. The ISI report was received with interest by people in Washington and at CINCPAC, but the Navy deemed it too extreme.

In January 1974 the Chief of Naval Operations submitted the Navy's plan of action for COTCO. The proposed COTCO plan, which ignored interactive message handling, called for improving communications in two phases. Phase one used existing equipment on the island, eliminating redundancy and saving some manpower; phase two would develop a rather large, sophisticated system to allow a single computer to handle messages for the whole island. This system was expected to save expenses because of economies of scale, but it presented no new approaches to the message-handling problem. The cost of the second phase was projected to be approximately $40 million over a period of three to five years, with implementation occurring in stages. The phase two plans only briefly mentioned a potential "conversational" mode during the latter years of developing that system, with the suggestion that some testing of the interactive mode take place. As submitted by the Navy, the COTCO report was criticized by many, particularly by CINCPAC, who was to be the primary user of the service, for being too expensive and offering no real improvement.

During this time, CINCPAC worked with DARPA to acquire a better understanding of interactive communication. CINCPAC obtained a few ARPANET terminals and began to understand the implications of an interactive message service. In early 1974 ISI, CINCPAC, DARPA, and the Navy began two years of discussion of automated message handling. All parties agreed that the plan described in the ISI report could never be practically implemented, but that conducting a test would demonstrate how this kind of service could work in an operational context and how effective it would be. Whereas the Navy had been viewing interactive message handling as something to try out after the main COTCO system was operational, CINCPAC began to view this sort of message service as the means to improve its communications. If COTCO was designed primarily to support the old style type of message handling, then its design would probably not include the equipment necessary to implement interactive message handling, possibly the dominant form of communications in the future.

Throughout 1974 and well into 1975, the COTCO plan was rewritten several times; each time the Joint Chiefs of Staff, CINCPAC, and others contributed to it, the concept of including an interactive test received greater emphasis. DARPA and the Navy began to talk about conducting a test with CINCPAC as the testbed. CINCPAC also took a stronger position; in August 1975, CINCPAC stated in a message [5] to the Director of Telecommunications and Command and Control Systems (DTACCS; formerly, Assistant Secretary of Defense for Telecommunications) that interactive message handling should be a part of any planned message-handling program. From January 1974 to August 1975, the Navy plan for COTCO was considered, revised, and reconsidered many times, until DTACCS disapproved the COTCO implementation on 20 August 1975. DTACCS permitted phase one of COTCO (the consolidation of services and elimination of redundancy) to be carried out, but cancelled the implementation of the proposed message-handling system. Instead, DARPA and the Navy were to conduct a test of interactive message handling, using CINCPAC as the test site. The Defense Communications Agency (DCA) headed a study to ensure that the test would take into account the needs outlined in the COTCO plan. DARPA and the Navy worked very closely with CINCPAC trying to define an appropriate test. In December 1975 DARPA, the Navy, and CINCPAC signed a Memorandum of Agreement defining the Military Message Experiment program [27].

1.1.2 The Initial System

The ISI team's report, issued in May 1973, indicated that the best way to consolidate telecommunications on Oahu would be to give all communications staff on the island access to VDTs connected to message-processing computers (i.e., put the staff *on-line*). There would need to be 5 message processing systems, 37 TIPs (Terminal Interface Message Processors, which link the terminals to the computers), and 2000 terminals, a conversion that would cost $22 million. The military viewed this approach as very radical, and ISI's recommendations never had a serious chance of being adopted. They did stimulate much interest in on-line interactive message handling, however, and this interest was channeled to Washington and CINCPAC. DARPA received enough positive feedback on the plan to encourage ISI to study the problem further.

ISI initiated the Information Automation (IA) Project, which was to investigate the kind of support required to implement an automated message service. The users of such a system would be naive about computers and would require a great deal of support from the computer software, even though the current state of the art was not yet adequate to contribute that support. Although intense research was not considered appropriate, the project would need to distill into a single system knowledge already available about providing a helpful and supportive user interface on a large scale.

The IA Project got under way in the fall of 1973. During its first year, during which the support requirements for this kind of system were studied, six reports were published [1, 15, 23, 32, 33, 42]. In addition to studying the computational possibilities, the project staff consulted with the Navy and with

CINCPAC to understand their message-handling needs and to educate them about on-line message-handling systems. ARPANET terminals were installed at CINCPAC and the Naval Electronics Systems Command to increase staff understanding of on-line service. Until then, the communications philosophy had always been to deliver the message as fast as possible: the faster it could be gotten out of storage the better, because communication was viewed as simply the delivery of messages. In contrast, on-line systems involve storing the message in a computer "mailbox" and letting the user come to it. Instead of the message being held in the computer for minutes, it would in fact be held there for months, to be filed and retrieved as desired. The military makes a very sharp distinction between communication (the delivery of messages) and administration (the storage and manipulation of messages after delivery). An interactive message-handling system blurs this distinction.

In the fall of 1974 the notion of conducting a test in an actual operational environment began to get some high-level DoD support. The Navy's interest in this test was related to COTCO; such a test was viewed as part of COTCO and was linked strongly to the $40 million program. In November ISI focused its design efforts on a message service for military applications, particularly for CINCPAC's environment. This service was documented in a report called the *Brown Bomb* [20], a design proposal to DARPA. While details of the proposed design changed almost entirely (little of the material on data structure, messages, and so forth survived), the basic architecture did prevail. This architecture is described more completely in Part 4 of the *SIGMA Final Report* (ISI/RR-81-95). The essential feature that distinguished the MME's message system from most of the message systems already on the ARPANET was the idea of a central database serviced by background processes, called daemons.[1] Instead of each addressee getting his own copy of a message, a citation would be delivered, pointing to the message and supplying summary information about it. If the user asked to read the message, the service would provide a copy from the central file, retaining the original for others' access. If the user changed his copy of the message, i.e., added a comment to it, the change would be submitted to a background daemon to update the central copy. The *Brown Bomb* strongly emphasized the design of a consistent supportive user interface; even though some of the particular aspects advocated in the early design were altered, the focus on supporting a computer-naive user prevailed throughout the program.

In February 1975, after DARPA essentially approved the design and gave the order to proceed, ISI began the implementation of what eventually became SIGMA, the message service designed for the MME by ISI. The original plan called for using the NLS[2] back-end system for the basis of functional performance. NLS was already being split into two parts, front-end and back-end, for the National Software Works: The user communicates with the *front-end*, which controls the display and communicates with the back-end; the *back-end* accomplishes tasks given it by the front-end and communicates directly with the computer. The idea was that the message itself would be manipulated and stored as NLS data; ISI would provide the front-end to that system. So ISI's work started by investigating NLS, ways to work with it, and developing SIGMA's front-end, including the command language processor, the editor, the tutor, and the user monitor, to support the user.

SIGMA was to be implemented in two phases. The service to prepare outgoing messages was given attention first because message preparation was the least developed part of then-current message services on the ARPANET and especially because of the requirements of the military *coordination* process. Coordination involves obtaining the approval of peers and superiors before releasing a message, since each message has the status of a formal document coming from the Commander-in-Chief of the organization. Nothing in the

---

[1] This approach was suggested by Albert Vezza of MIT.

[2] NLS (oN-Line System) was developed at the Stanford Research Institute in the mid-60s and was one of the first office-automation systems

available ARPANET message services adequately handled the coordination process, so this part of the system would have to be designed from scratch. It was expected that this first phase would be complete in 1975; the second phase, implementing an appropriate way to handle incoming messages, would be complete by October 1976. After a period of integration, in January 1977 SIGMA would be ready for testing with some (hand-picked) users. By July of 1977, SIGMA would be ready for CINCPAC users to begin operational testing.

In the spring of 1975, ISI produced a document for DARPA proposing the format for a test plan of an experiment involving automated message handling. MITRE was chosen to administer the test, so ISI's proposal was shipped to MITRE for consideration. By September 1975, MITRE had a full test plan draft, which was approved in final form a year later.

By the end of the summer of 1975, negotiations between DARPA, the Navy, and CINCPAC for conducting a test looked very promising. ISI's work with NLS showed it to be so intractable that the plan for using it was dropped; ISI would implement new message-access mechanisms. DARPA began to emphasize the development of the new system. MITRE was put in charge of the message service security. In order to assure that some system would be available for the experiment, DARPA asked BBN and MIT to develop alternate message services for the test.

Early notions of the CINCPAC experiment had the message-processing computer housed at a Navy installation in the continental United States; CINCPAC was to be connected to the computer by the ARPANET. Because of network delays, ISI advocated reprogramming the terminal's microcode to make it intelligent; it was felt that response time would then be acceptable even though the host was 2000 miles away. Indeed, the direction of ISI's project began to assume this type of architecture.

By the fall of 1975, SIGMA was working well enough to permit actual demonstration of the simulated terminal interface, the language processor, the message editor, and the limited delivery facility. In early December the Memorandum of Agreement was signed between DARPA, CINCPAC, Naval Electronics Systems Command, and the Naval Telecommunications Command. The signing of this agreement deeply committed DARPA to provide a useful service. In January 1976 a meeting was held at NAVTELCOM facilities in Washington with DARPA, the Navy, MITRE and the three contractors: ISI, BBN, and MIT. At this meeting discussion centered upon a plan for the three contractors to work together to make the test successful. Each of the contractors had its own proposed message service; it was agreed that there was no way to combine the three systems for a single test because each was completely different from the others, so the program became a three-way competition.

The three contractors concurred that the user should communicate with the message service through a VDT. ISI's proposal for putting intelligence in the terminal, to provide multiple windows and two-dimensional editing capability, was adopted; all agreed to use the ISI terminal. BBN later chose to stay with a more conventional VDT.

Detailed plans for handling the MME itself were taking shape in the spring of 1976. Contracts were let to CTEC and MITRE. CTEC would provide the test director, who would be independent of the implementors. MITRE had developed the test plan, which called for the message system to collect data on how it was used. The Navy planned to run the computer in Cheltenham, Maryland, near Washington D.C., and to start the test in March 1977. After the dissolution of the COTCO plan, the Navy had incorporated the MME test into a larger program called DISTAN (Distributed Interactive Secure Telecommunications Area Network). The DISTAN program never went much further than MME; the Navy eventually dropped DISTAN.

An ISI staff member developed a technique called a Protocol Analysis Test, which comprised an extended interview and simulation with candidate users to learn how potential users felt about an automated message service, and to gain information about design, data format, functionality, vocabulary, and so forth. In May 1976, a test was conducted with Navy personnel in the Washington D.C. area to evaluate the utility of protocol analysis. The results of the May interviews [16] justified a similar test on the island. In July, an ISI team interviewed 24 CINCPAC users, collecting information on what an interactive message service could do for them, how they could use it, and their preferred user interface. That information was used in ISI's design. The report on that protocol analysis test appeared as a working paper in September of 1976 [17].

The original plan to locate the computer in Cheltenham, Maryland, was abandoned in 1976. Reduced performance, potential unreliability, added expense, and CINCPAC's concern at having its messages appear at the Navy site halfway around the world contributed to this decision. Instead, CINCPAC agreed to provide room for the MME computer in the Command Center building, next to the WWMCCS (World-Wide Military Command and Control System) computer. The plan thus changed from remote to local computer access. Although CINCPAC felt its WWMCCS operation staff could support the MME computer, some people with TENEX expertise were required to manage it. BBN was chosen to manage the operation and maintenance of the hardware.

As 1976 drew to a close, a choice had to be made among the three message services. The decision was to conduct a run-off in February 1977 to evaluate the three different services, and select which one would go on the island. The other two were to be used at other sites, in structured tests. (These structured tests never occurred.)

BBN was the site chosen for the evaluation; the evaluators were from CTEC, NAVELEX, CINCPAC, the Naval Research Lab, and MITRE. The group was chaired by the MME program manager from DARPA. Each message service was given a three-day evaluation. ISI sent seven terminals to BBN for use in the evaluation. Each group had one day to prepare for the evaluation and one day to introduce and present its message service to the evaluation team and to work with them on how to use it. On the third day, the evaluation team worked on its own to judge the message service's effectiveness.

In early March the evaluation team went to Washington to pool their opinions and make a selection. Each message service was evaluated for functionality, security, and user interface. ISI's SIGMA was ranked the highest in each category: its user interface was suited to the military, it had appropriate provisions for security, and it provided functions that paralleled existing manual practice. SIGMA's deficiencies were mainly in functionality and performance: it ran at such a slow rate that only three users could work reasonably well. The decision was made to select SIGMA on the condition that ISI would eliminate SIGMA's deficiencies.

ISI devised a plan for doing so, the SIGMA Transition and Deficiency Amelioration Plan [37], which called for installing SIGMA in May 1977 and allowing "friendly" users to shake down the system. New software releases were scheduled every month from May through December; in December, the experiment itself would begin with the final software release, and "real" users would be put on the system. During this period some training and on-site testing would occur, connections would be made to LDMX, security evaluated, and so on. In effect, ISI had eight months to make the system work, with the test to begin in January of 1978. The test was scheduled to run through June 1979, six months less than originally planned.

SIGMA's poor performance was only one of the deficiencies noted in the Deficiency Amelioration Plan. Others included on-line lessons and exercises, considered essential for training at CINCPAC. Some utilities to support the operation needed to be developed, such as a robust terminal protocol to allow for communication errors and a data collection package to obtain information on the system's use. SIGMA had to be extended to handle printers and to be coupled to the LDMX in order to receive messages. Some

functional features still had to be incorporated into SIGMA: an archive system, an alert mechanism to indicate the arrival of new messages, and modification of access control. Both routing of incoming messages and procedures for handling outgoing messages had to be improved. In addition, an entire collection of general improvements still had to be worked out.

The performance goals of this transition plan were to support 5 users on the system in June, 10 users in September, and 25 users in December, a five-to-one increase in throughput within six months--a very significant improvement. SIGMA would have to be overhauled considerably to be able to support this level of performance.

In May 1977, a PDP-10 with a KA processor was installed at CINCPAC with 256K of core, only half of what was planned. SIGMA was installed on the computer, and five terminals were delivered so that initial system testing could begin. The equipment included two processors, two channel adaptors, two network interfaces, two pagers, a single bank of memory, and a single disk drive and disk interface. Processors were duplicated presumably for reliability; if one was down, the other would be available. Ironically, the most fragile components, the disks and the memory, were the only pieces of equipment not duplicated.

1.1.3 The Evolution of SIGMA

From June through August 1977, ISI followed its transition plan fairly well, each month delivering a release with new improvements directed toward functionality and improved performance. Project members were also involved in security tests, installing a remote link to the block house, fixing bugs in SIGMA code, coping with TENEX problems, testing new software, and making on-site visits to CINCPAC to assist in the operation of SIGMA. ISI hired an on-site representative who took up residence at CINCPAC in late August. Unfortunately, most performance improvements achieved during this time were counterbalanced by the added functions, which tended to degrade performance.

In the summer of 1977, a second 256K of core was added, the single most important step toward significantly improving performance up to this point. In October, when the goal had been to support ten simultaneous users, only about five users could be supported.

Performance became the critical item in October, since the promised level had not been met. The schedule was completely revised, with all the functions promised for the October, November, and December releases postponed. It was decided that the project should focus strictly on performance; in December an evaluation would be made of whether SIGMA's performance would ever be adequate to support the Military Message Experiment goals. The functions promised for those fall releases were deferred until after January. The November release (release 1.7), the critical one for performance, was deferred until early December to allow extra time to make the necessary alterations.

TENEX itself seemed a likely source for performance improvements. Don Allen from BBN provided some tools to measure the CPU's use of time and to take program counter (PC) samples. These tools were very helpful in determining why the computer's response was so poor and how to improve it. The results not only indicated that most of the planned changes for improving performance were correct, but also helped to generate several new ideas for making improvements.

The basic problem was insufficient processor and memory. TENEX is set up to provide many general capabilities that can be invoked from the user's code. It turned out that more than 75 percent of SIGMA's computing cycles were being spent in TENEX code, so it was necessary both to alter SIGMA to make it rely

less heavily on TENEX functions and, where possible, to improve those TENEX functions. One slow TENEX function, the String Output (SOUT) JSYS, was redesigned by BBN, cutting the use of computer cycles 80 percent. Some of the changes made in SIGMA to improve performance went very deep in the representation of data, permeating virtually every module, so they were difficult and time-consuming to implement. After these changes had been completed, numerous bugs had to be eliminated. However, results were promising: the amount of computer processing necessary to display messages, for instance, dropped from 6.3 seconds of computer time to 2.3, better than a two-to-one improvement. There were improvements in other commands too; for example, DISPLAY FILE was reduced by 50 percent. By the time of release 1.7, performance had almost doubled. Thus, in December, SIGMA could comfortably support about ten users. Although the target had been to support 25 users by December, even with the completed changes SIGMA could not yet support half that number. There were still some other changes to improve response, but it seemed doubtful that anything planned would provide another doubling.

Another approach seemed promising. The MME was using only one of the two available processors. The other sat idle for backup. If SIGMA's load were distributed between the two processors, it was felt performance would almost double. Configuring the two processors to operate in tandem did not seem to be too difficult; SUMEX at Stanford operates its PDP-10 this way normally, running two KI processors together. A consultant from SUMEX was asked to determine the feasibility of linking the two KAs at Camp Smith. He indicated that connecting the two processors would be fairly easy and would provide about an 80 percent increase in computing power. With other changes to SIGMA, this would allow it to handle 25 users. BBN, however, who was operating the computer system, estimated the price to be approximately $600,000 for the changeover to a dual processor system.

If $600,000 were to be spent, it made more sense to purchase a new DEC KL processor, which would increase performance by a factor of three. ISI proposed this possibility, arguing that the easiest and cheapest way to increase computing power was to enhance the hardware at Camp Smith. In the long run, purchasing the KL would be cheaper than trying to force the SIGMA software to be more efficient; with the KL, there would be an immediate and obvious gain in computer cycles. Once the experiment was over, the KL could be used for another project. In general, the KL would provide more substantial results for the experiment, results that the MME sponsors were anxious to see.

At the end of 1977, it was decided to acquire a new KL processor and continue with SIGMA at CINCPAC. Operating TENEX with WWMCCS operators was not working well, so the plan also included having ISI take over operations with a full, round-the-clock operations team, system programmers, and maintenance crew. The experiment would have to be shaped differently to accommodate the now increased lead time before proceeding with a full set of users.

Early in 1978, the ISI design team and the rest of the staff involved in MME re-evaluated the progress of the experiment and planned for the future. Much time had been devoted to providing functions on SIGMA while trying to improve performance, but the fundamental areas of reliability and operability had been neglected. For example, the daemons badly needed reworking, both to simplify them and to provide better controls from the operator's point of view. Better controls were also needed for error situations and for logging appropriate diagnostic data. The focus therefore turned to the serious problem of reliability. The functionality enhancements were scheduled for gradual introduction; all would not be completed until after the new machine arrived and performance increased.

SIGMA's reliability problems were compounded by reliability problems with TENEX. BBN had lost the one programmer it had on the island and had had to run the computer without any real systems programming expertise. In March, BBN replaced the systems programmer with the man who had written the original

TENEX-LDMX interface code for TENEX. He was able to strengthen that code and consolidate the entire operator monitoring system. ISI was rewriting the daemons and improving some of the functions so that they could be ready for release 2.0 in June. Releases 1.71, 1.72, and 1.73 delivered during this time fixed various bugs.

About this time the Navy appointed a Blue Ribbon Committee to review the entire MME program to determine whether the Navy should continue its support. This committee spent about a month examining the SIGMA system and the installation at CINCPAC, interviewing the users and reviewing the program plans. They concluded that although MME had experienced delays and system performance was inadequate, the program had reasonable solutions under way and it should be given continuing Navy support.

By May 1978, the system was stabilizing well enough to permit messages to be taken from the LDMX on a continuous basis. The daemons in SIGMA, though still the old version, were sufficiently reliable for storing messages. Prior to this time, messages were coming in sporadically and there was no guarantee that all messages were arriving.

SIGMA release 2.0, in June of 1978, had new daemons, archive facilities, and functional enhancements. The resulting improvements in the operability and reliability of SIGMA were very dramatic. Performance also increased, since the new daemons were smaller, simpler, and required less resources to run. In general, although the hoped-for number of users could not yet be supported, there was some semblance of stable operation. July brought the decision to begin Limited Experimental Use, which allowed selected people from CINCPAC Operations (J3) to use the system experimentally. These selected users would recommend improvements to be incorporated into subsequent software releases.

Release 2.1 in September offered functional enhancements and performance improvements. Some plans were made to install terminals at several other sites on Oahu (for example, the offices of the Commander-in-Chief, Pacific Fleet and the Pacific Air Forces), but because of the already numerous tasks pressing on the staff, it was decided not to extend the service beyond the CINCPAC Command Center. Extending the service outside the Command Center would have provided some interesting data on use of the system (direct message communication to these outside organizations would have been a truly new capability). The extension of service would have stressed access control mechanisms and offered an opportunity to study the practicality of an informal interorganization message service.

When the KL processor with 1024K of memory was installed in October, most of the problems with SIGMA's performance vanished. Although the computer resources available even with the KL were not sufficient for extending the service, they would support a meaningful test of SIGMA at CINCPAC. However, as with any new piece of hardware, the KL introduced some problems. Time was needed to shake out the bugs and get the processor working smoothly. The new operations crew needed time to work out procedures and learn to work as a team. During October and November, the computer ran with only a few difficulties, but in December severe disk problems brought a reduction in SIGMA's availability. Uptime dropped from 95 to 86 percent. In addition, the entire disk file was destroyed and had to be restored from backup tape.

Release 2.2 was installed in January 1979 with functional improvements such as an Alert facility, Readdressal, and better coordination features. With this release CINCPAC authorized the use of SIGMA for outgoing messages to AUTODIN. After release 2.2, disk problems continued on and off. Uptime in January was 93 percent, but in February only 83 percent. Despite the lack of reliability, Full Experimental Use began in February. Users were encouraged to do their work with SIGMA and to take advantage of the capabilities offered. SIGMA had almost all of the desired functions. Some problems with the outgoing message capability were discovered during tests in January and were fixed in releases 2.21, 2.22, and 2.23, as were some operational aspects.

In March, a bad filter was removed from the main power system. The filter was part of the equipment maintained by the CINCPAC community and had probably been faulty for some time. After the bad filter was removed, the system reliability increased dramatically. The recurrent disk troubles were also repaired. By the end of this month, the system reliability improved to 96 percent.

Early March was the time for the full-scale Exercise Power Play, a global exercise involving U.S. armed forces centered primarily in Europe, which made CINCPAC's involvement considerably less than it would have been if the exercise had been in the Pacific theater. The 24-hour watch team of Command Center personnel conducting the exercise were free to use SIGMA to see how it would be helpful in a crisis, although no one depended on SIGMA exclusively because of its reliability problems. Message handling on paper and on SIGMA proceeded in parallel, involving double work for some of the staff.

The faulty power filter was being replaced at this time, so SIGMA was unavailable for significant periods of the exercise. Questionnaires distributed to members of the exercise watch team showed that, although they felt it had promise, they were not able to work with SIGMA enough to evaluate it adequately.

During the succeeding months, reliability continued to improve, but the system was still subject to occasional fits of catastrophic failure. To store messages, SIGMA had depended on a single density disk, which ran very full and could hold only two weeks of message traffic. In May a double density disk replaced the old one, allowing 30 days of message traffic to be accommodated. Users were much more satisfied with the increased storage; they could access more messages on-line and less often had to wait for their retrieval from archive. In addition, system crashes resulting from overrun of disk space, a fairly common occurrence, were eliminated. Users became more inclined to use SIGMA as they gained confidence in its reliability.

Unfortunately, these improvements in reliability happened too late. The decision was made in the spring of 1979 that the system would be removed from CINCPAC at the conclusion of the experiment, 1 October 1979. This decision was based primarily on the system's lack of reliability and that service could not be extended throughout more of CINCPAC. The decision to remove SIGMA disappointed the staff of the MMF and many of the users who had become accustomed to working on the message service.

The last major SIGMA release, 2.3, was made in early June, although a few minor releases to fix bugs did come out before the end of the experiment. Release 2.3 contained a collection of functional improvements, many of which were responses to user requests, such as sorting files and highlighting messages. SIGMA continued to be used during the summer months, even though it was scheduled for removal. New staff members at Camp Smith, unaware of SIGMA's earlier shaky reputation, used SIGMA and preferred it to the paper processing of messages. These new users of SIGMA seemed to believe that the system was worth keeping. With both a new CINCPAC and a new J3 at Camp Smith, an attempt was made to reverse the decision to remove SIGMA. CINCPAC notified the Joint Chiefs of Staff that he had re-evaluated SIGMA and that SIGMA's much improved reliability and the staff's better understanding of the system's functional capabilities justified keeping SIGMA. The appeal came too late; the system was removed as planned.

During a two-week period in August before SIGMA was dismantled, users were asked to work with SIGMA to send outgoing messages so that evaluators could obtain a better understanding of this capability. Detailed results of this evaluation are given in MITRE's reports [13, 14]. During September, the last month of the system's operation, Exercise Power Play was re-run at CINCPAC in order to better evaluate the system's utility in a crisis. All the messages received in the previous exercise were retrieved, edited back to their original state, and then re-entered into the system as new messages. An operator controlled the introduction of the messages so the pace could be varied. The exercise was conducted essentially as it had been in March, except this time SIGMA was the primary message-handling medium, and all messages, incoming and outgoing, were kept within SIGMA.

Messages were first introduced at a rate roughly equivalent to real time, and then as the users warmed up, the message load increased. For the last six to eight hours, messages were submitted about every two minutes, a fairly heavy load, although not a saturation point for the paper system. SIGMA performed very well; the users found it worthwhile. However, the results were inconclusive in determining if an automated system is more effective than a manual system in a crisis. More discussion of these results is presented in the official MME Final Report [29].

On October 1, 1979, the Military Message Experiment was completed. The users were allowed time to take files off-line and convert them to paper form. The system was then shut off and sent back to ISI, where it has since been added to the ARPANET. This was the official end of the operational part of the MME. The remaining part of the MME is the documentation of its results, of which this report is one contribution. A formal final report is being prepared by the Naval Research Laboratory.

1.1.4 SIGMA Staff

Although the experiment took place at Camp Smith, Hawaii, many people supporting the experiment were not at CINCPAC itself. The largest group, of course, was ISI's Information Automation (IA) Project staff, which ranged from 6 to 10 people over the five-year history of the project. The IA Project developed the SIGMA software (the computer program) and the terminal firmware (modifications to the operating software for the VDT's standard microprocessor) used in the experiment, and was involved in all aspects of the service: training [18, 34], security, and operation.

1.2 SUPPORTING DOCUMENTS

This volume of the MME Final Report is written by the staff of the the Information Automation project at ISI and reflects only the experience and views of the experiment from the perspective of the software developer. It is ISI's contribution--Volume V--to the full MME Final Report being compiled by the Naval Research Laboratory. At the time of this writing the full official MME Final Report is not complete; however, the volumes to be included are:

Volume	
I	Executive Summary
II	Final Report
III	User View
IV	Message System Utility
V	SIGMA Final Report
VI	Data Analysis and Discussion
VII	Training

There are a number of other documents concerning the MME from which one can obtain a wider view of the experiment. Of particular interest are two preliminary reports published by the Naval Research Laboratory. The MME Quick Look Report [46] describes the progress of the experiment from May 1977 to November 1978. Although there is limited data analysis presented, excerpts from interviews with the users provide insight into how CINCPAC viewed the early experience with automated message handling. A similar report [22] describes the conduct of the experiment from November 1978 through March 1979. Some preliminary conclusions are drawn based on some early data analysis, observations of the use of SIGMA, and more user interviews.

Copies of the Quick Look Report, Mid Experiment Report, and all volumes of the Final Report, as they become available, may be obtained from:

```
Naval Research Laboratory
Washington, D.C. 20375
Attn: Code 7503.
```

The bibliography of this report contains a collection of other documents that are relevant to the MME. The more pertinent of these are cited in the text of this document.

Part 2 of the *SIGMA Final Report* (ISI/RR-81-94) describes the functionality of SIGMA as a user views it. Part 3 (ISI/RR-81-94) contains opinions of the authors (all members of the ISI team that developed SIGMA) formed from a review of data abstracted from user interviews, discussions with users, and other peripheral observations. Part 4 (ISI/RR-81-95) describes the design of SIGMA in fair detail. There is more detailed information about the design of SIGMA stored in the collection of TENEX directories that make up the SIGMA program at ISI. The most extensive documentation is the source code itself, which is well commented. The TENEX directories constituting SIGMA are listed below:

CINCPAC	Utility programs and data that are local to CINCPAC
CMP-DAEMONS	Configuration Management programs, runfiles, and documentation (i.e., SIGMA release procedure)
DAEMON-LOCAL-STATE	Support files for the operation of the daemons (e.g., Error-logs, Trace files, Queues, Message Directories)
HP-MME	Programs to generate Terminal firmware and firmware Sources
IA-ARCHIVE	Archive daemon Source code (development directory; production directory is MME-ARCHIVE)
IA-BATCH	Utility programs and files for the Batch Processor, used in the development of SIGMA
IA-COMMON	Source code for packages common to all SIGMA (e.g., error package, tools to define micros, multiprint package) (development directory; production directory is MME-COMMON)
IA-CITATION	Citation daemon Source code (development directory; production directory is MME-CITATION)
IA-FM	Functional Module Source code (development directory; production directory is MME-FM)
IA-FOLDER	Folder daemon Source code (development directory; production directory is MME-FOLDER)
IA-GENERAL	Source code for support of runtime SIGMA (JSYS package for interfacing

	to TENEX, binding specifications, pseudo-interrupts, facilities to effect sharing of common code between forks, etc.) (development directory; production directory is MME-GENERAL)
IA-HELP	Source code and text for the Help and Tutor systems (development directory; production directory is MME-HELP)
IA-MESSAGE	Message daemon Source code (development directory; production directory is MME-MESSAGE)
IA-OTHER	Facmod and Msgmod Source code (development directory; production directory is MME-OTHER)
IA-RECEPTION	Reception daemon Source code (development directory; production directory is MME-RECEPTION)
IA-RUNTIME	The SIGMA running code and support programs running code (i.e., object code) (development directory; production directory is MME-RUNTIME)
IA-SIGMA	Command Language processor Source code and Command Table (development directory; production directory is MME-SIGMA)
IA-SUPERSTRUCTURE	Source code for programs common to daemons (e.g, CCP, PC) (development directory; production directory is MME-SUPERSTRUCTURE)
IA-TERMINAL	Terminal Driver Source code (development directory; production directory is MME-TERMINAL)
IA-TEST	Files to support the testing of SIGMA (e.g., test scenarios, test messages)
MME-UTILITIES	Source code for most utility programs (e.g.. FCHECK, MSCAN, FLAGS) and release procedures for User Job
SIGMA-DOCUMENTATION	Documentation files for various pieces of SIGMA
SIGMA-LOCAL-STATE	Files used by SIGMA during operation

PART TWO:

FUNCTIONAL DESCRIPTION

2.1 INTRODUCTION

This part of the *SIGMA Final Report* describes the functionality of SIGMA as a user views it. Before continuing here, the reader might turn to the appendix of this part (following p. 2-35) to review "SIGMA--An interactive message service for the Military Message Experiment" [38], a paper given at the 1979 National Computer Conference, which gives an overview of SIGMA. In addition, it may be helpful to glance through the *SIGMA Reference Manual*, which contains instruction formats, a level of detail not presented in this document.

This part introduces the reader to the system in roughly the sequence that a new user is exposed to it. It starts with a discussion of the terminal, followed by the log on procedure, then the various objects the user deals with in SIGMA, and the operations he may perform on them.

2.2 THE TERMINAL

The *HP/MME terminal*, a special terminal developed for use with SIGMA, consists of a Hewlett-Packard 2649A terminal with minor physical modifications and special firmware developed by ISI [39]. It contains 12K of display memory, but at any time only 1920 characters may be presented on-screen (24 lines of 80 characters). Figure 2-1 illustrates the MME terminal. To the right of the CRT are 4 LED lights labeled *TOP SECRET*, *SECRET*, *CONFIDENTIAL* and *UNCLASSIFIED*. They indicate the highest classification of data on the screen. As a user views objects (e.g., files, messages) at different security levels, SIGMA automatically changes these lights.

The keyboard unit is connected to the main terminal by cable so the user may adjust its position for maximum comfort. The keyboard contains a standard typewriter keyset, a cluster of 23 additional keys to the right, and a set of 26 function keys laid out above. The typewriter keyset is used for generating text. Fourteen tan keys on the right side produce actions local to the terminal, such as moving the cursor and scrolling the screen. The remaining keys (blue key caps on the right and the upper keyset) are Function keys. They send a request to SIGMA to perform some operation. The specific function each key requests is labeled on the key cap for the blue keys, or above and below the key on a plastic keyboard overlay for the upper keyset. Functions whose labels are above the keys are activated by holding down the shift key while they are pushed. The two keys on the left end of the upper keyset (!!*RESET*!! and !!*ONLINE*!!) are not function keys; their special role is described in the next section.[3] Seven LED lights show through the function key overlay. Only the four centered in the yellow background area labeled "CURSOR SECURITY LEVEL" are meaningful in SIGMA. These are similar to the screen security lights: there is one for each classification. These indicate the level of the object where the cursor resides. The cursor indicates the position where data typed on the keyboard will be entered. SIGMA defines different areas of the screen, called windows, for different purposes; these windows can be at different security levels (described in section 2.10, page 2-13). If the cursor is in a Secret window, any data typed is considered Secret and will be so treated by SIGMA. Thus the keyboard security lights indicate the classification at which typed data will be entered.

---

[3] In this document, function key operations are shown bracketed with double exclamation points (e.g., !!*EXECUTE*!!).

Figure 2-1: The MME Terminal

2.3 LOG ON

As a user approaches an MME terminal to start up SIGMA, he must first determine if the terminal is already in use. If the screen is blank (power off) or if it contains a single line of text saying

```
TERMINAL READY
```

or

```
TERMINAL FREE - previous user has
logged off
```

the terminal is free. If the terminal has a normal SIGMA presentation on screen, someone is already logged on that terminal, and it cannot be used until he has logged off.

Before trying to start SIGMA, the user must place the terminal's !!*ONLINE*!! switch in the depressed position (on-line). If the terminal power is off, turning it on is sufficient to cause SIGMA to start. If power is already on and the terminal is free, holding down the !!*CNTL*!! key and pushing !!*RESET*!! will initiate a SIGMA job. An alternative procedure is to push the !!*ESC*!! key (see section 3.3.2.1, page 3-12, for a discussion of startup procedures).

SIGMA will respond to a startup request by presenting a "log on" template, as shown in Figure 2-2. The user fills in this template by moving the cursor to each empty field and typing in the appropriate information. Text editing features in the terminal facilitate filling in the form (see section 2.8, page 2-9, for a description of these features).

```
Logon name:
Logon password:
Max. Sec. Level:
[If required enter] Identity:
ID password:
```

Figure 2-2: Log on Template

On the first line of the form, the user enters the name of his office code. On the second line the user enters the appropriate office password; the password itself is not displayed. The third line requests the maximum security level the user intends to use for the session. SIGMA inhibits a user from accessing objects that are classified higher than the current session's security level. The remaining two lines of the template are for the user's name and password.

This two-level log on procedure, office code and personal name, provides accountability. Each action that SIGMA records, such as approving or releasing a message, contains the name of the office taking the action and the individual who was acting in that role, even though only the office code shows on the message itself. This allows secretaries or administrative assistants to act for their superiors (they first need the office password), while keeping them accountable for their actions. A user may log on with his personal name directly, without an office code, but he then cannot directly access objects belonging to the office or act for that office. SIGMA will not allow a user to log on with an office code without a personal name (referred to in SIGMA as his *identity*).

When the user presses !!*EXECUTE*!!, SIGMA processes the log on request. If the passwords are correct and the maximum security requested does not exceed a predefined maximum security for that user (office code and identity) or that terminal, the job is started up and the log on phase is finished.

If the password for either the office code or identity is incorrect, SIGMA will reject the log on and produce an error message telling the user the password is incorrect. As a security feature specified by CINCPAC, the third time a user enters the wrong password SIGMA locks out that terminal. The System Security Officer (SSO) must then intervene to unlock the terminal (see section 4.14.4, page 4-106 for a description of the SSO program).

A user has 6 minutes to fill in a log on template. If he has not successfully logged on in that time, SIGMA clears the screen and logs off the TENEX job. This housekeeping keeps the system clear of not-logged-on jobs.

2.4 STANDARD SCREEN

Once log on is complete, SIGMA presents its standard screen format to the user. The top line, called the *Alert line*, contains status information about the system and the job, including the SIGMA release number (what version of the system is running), the "load average" (a figure indicating how heavily the computer is

loaded), the day of the week, the date, local time and Zulu (Greenwich Mean) time. This is also where SIGMA notifies the user of the arrival of new messages and "alerts" (see section 2.16, page 2-33).

The *Status line* is the second line on the screen. It initially contains just the log on security level and office code. As the user "opens" objects in the message service, the name, security level, and the type of each object will also be presented on the Status line.

The third line is the *Feedback line*. It provides dynamic feedback from SIGMA to the user as various operations proceed. For instance, before log on it contains the message

```
**Please enter
the necessary information to log on.
```

During execution of instructions it may say something like

```
***clear view* being processed
```

to indicate it has accepted the !!*CLEAR VIEW*!! instruction and has started. If an error condition is encountered, the user is informed on this line.

These top three status lines are continually updated. For example, the time is changed every minute on the Alert line. The terminal inhibits the user from putting his cursor into these lines; he can never alter their contents.

Lines 4 and 5 are permanently assigned as the *Instruction window*. This is the area of the screen where the user types instructions to the system. Instruction entry is discussed in more detail below. The beginning of the Instruction window is marked by two bell-shaped characters, which in this document are shown as ††. If the user types more than can fit on one line, the terminal automatically wraps onto the next line. If the instruction spills onto a third line, the Instruction window automatically scrolls so the first line disappears, the second line appears on line 4 and the new third line is on line 5. Instructions rarely require this much space.

The rest of the screen is available for working space, where messages, files, and other SIGMA objects are displayed and edited. This area may be occupied entirely by a Display window, entirely by a View window, or, in split-screen mode, by both. An object that is "opened" for editing is shown in the Display window. An object that is only expected to be referenced and not altered is normally shown in the View window. For instance, when a user has completed Log On, he is presented the System News in the View window, which occupies the full working area (19 lines) since there are no open objects to present in the Display window. The news, entered by the System Control Officer, contains general information relevant to SIGMA users, such as scheduled down times, new features, or new procedures. The split-screen mode is used when a user is working on one object in the Display window and wishes to reference some other information. In split-screen mode the View window occupies the last 9 lines of the screen and is shown in lower intensity to distinguish it from the Display window. Section 2.9.1 on page 2-11 describes how the user controls these windows.

2.5 INSTRUCTION ENTRY

Most typed instructions require that one or more parameters be specified. For example, the **DISPLAY FILE** operation requires the user to specify the name of the file to be **DISPLAY**ed,[4] while **CREATE FILE** needs the name of the file and its security level. SIGMA requires that the name of the instruction be typed first in the Instruction window, followed by the appropriate parameters in any order. When the instruction has been properly entered, the user depresses the !!*EXECUTE*!! key to tell SIGMA to act. Pressing !!*EXECUTE*!! automatically turns off the cursor and disables the keyboard. When SIGMA has taken action on the typed instruction, the cursor is returned, and the keyboard is enabled.

SIGMA's first step is to try to interpret what has been typed as an instruction. As an aid to the user, SIGMA will accept partially completed or improperly spelled instruction names and parameters. It will expand and correct them to the degree it is able. The algorithm used in this process requires that the first letter of the instruction be correct. The expanded instruction is parsed and presented in the Instruction window in place of what the user originally typed. If the instruction is complete and unambiguous, the user is told on the Feedback line

```
**Please confirm.
```

When he confirms by pushing !!*EXECUTE*!! a second time, SIGMA proceeds to execute the instruction.

If the parsed instruction is incomplete or ambiguous (there is more than one possible interpretation), SIGMA reports

```
**Your instruction is ambiguous.  Use PROMPT for more information.
```

on the Feedback line. The user may use the PROMPT facility (see section 2.7.1, page 2-6), the HELP facility (see section 2.7.2, page 2-7), the *SIGMA Reference Manual*, or he may ask a friend for guidance. The instruction, as parsed, stays in the instruction window, where it may be edited or erased and a new instruction entered. The next time the user pushes !!*EXECUTE*!!, SIGMA will attempt to interpret the instruction from the beginning again.

After a user becomes experienced with SIGMA, he learns the minimum he must type to get his instructions properly parsed. At that point having to confirm each instruction becomes annoying. The user may then ask the System Security Officer to make him an *intermediate user*. Intermediate users are asked only to confirm instructions that will permanently affect the database, such as **DELETE FILE** or **LOG OFF**.

Sometimes intermediate users are unsure whether the instruction they have entered will be correctly interpreted. The function key !!*EXPAND*!! causes SIGMA to parse what is entered in the Instruction window and display the expanded form, but not execute it. The user then presses !!*EXECUTE*!! if he likes it, or alters the expanded instruction if he does not.

At any time during the instruction entry, if the user wishes to cancel the instruction being entered, he can use the !!*CANCEL*!! function key. This key is effective until the final confirming !!*EXECUTE*!! is entered. !!*CANCEL*!! clears the Instruction window, and leaves it ready to accept fresh instruction.

---

[4] The names of SIGMA instruction are indicated by **BOLD CAPITALS** in this document

2.6 FUNCTION KEYS

The instructions most often used have been assigned function keys. Function key operations do not require confirmation and do not take typed arguments, although some functions use the cursor position or the location of special marks called !!*HERE*!!s (see section 2.8, page 2-9). Function keys act like the !!*EXECUTE*!! key; they also lock the terminal keyboard and turn off the cursor. !!*EXECUTE*!! is actually just a particular function key which causes SIGMA to parse and execute what has been typed into the Instruction window.

2.7 USER ASSISTANCE

There are three sources of user assistance in SIGMA: Prompt, Help, and the Tutor. The paper "On-line tutorials and documentation for the SIGMA Message Service" [35] describes these mechanisms in more detail. The user's view of these facilities is abstracted from that paper and presented here.

2.7.1 Prompt

Prompt is a limited aid to the user for entry of instructions. When the user presses the key labeled !!*PROMPT*!!, SIGMA presents all instructions that are legal in the current state, considering what has been typed in the Instruction window. If nothing has been typed, he will see all the instructions valid in the current situation. If the letter C has been typed, he will see all the forms of all instructions beginning with C. When the user presses !!*PROMPT*!! again, the original state of the screen will be returned. If the user types an ambiguous instruction in the Instruction window and tries to execute it, SIGMA will respond with an error message in the Feedback line telling the user that the instruction is ambiguous and suggesting he press !!*PROMPT*!!. If the user then presses the !!*PROMPT*!! key, SIGMA shows prompting for just those instructions that it still considers possible candidates for the instruction intended, based on what has been typed in the Instruction window. Instructions that SIGMA interprets as "most likely candidates" (which most closely match the number and types of the given parameters) are shown highlighted.

For example, suppose the user has a file named Pending and a text-object named Papa. If he types

```
††D P
```

in the Instruction window and pushes !!*PROMPT*!!, the Prompt facility will show four possible instruction interpretations:

```
Display Text <Existing Text Name>
Delete  Text <Existing Text Name>
Display File <Existing File Name> <Security>
Delete  File <Existing File Name>
```

With each instruction is shown a brief description of its function and a stylized form of the instruction with its parameters.

Once an instruction (or list of instructions) is shown by Prompt, the user can select one of them by placing the cursor on it and pressing !!*PROMPT*!! again; this expands the description of that particular instruction, showing the syntax and meaning of each of its parameters.

While viewing a Prompt display, the user may edit the Instruction window. Hitting !!*EXECUTE*!! will cause SIGMA to restore the original screen and attempt to execute the edited instruction. The screen will be returned to its original state if the user hits !!*PROMPT*!! with the cursor in the Instruction window. !!*CANCEL*!! will also return the screen to normal, but the Instruction window will be cleared.

2.7.2 Help

If the user needs a more detailed description of an instruction or cannot remember which instruction to use, he can request the next level of documentation: Help.

2.7.2.1 Selectable terms

The Help system provides documentation by describing *Terms* (names of instructions, facilities and concepts relevant to SIGMA). The user can request documentation of the available Terms through the Help processor's flexible mechanisms. The upper window of the Help display appears as

```
HELP   SERVICE FACILITIES   Current Term: <some key phrase>
BACK   FORWARD              New Term:
```

The *Current Term* field shows the Term which is currently displayed (e.g., "<some key phrase>" in the example above).

The *New Term* field allows the user to type in a new Term for which he wants Help. Spelling correction is provided by means of the same algorithm employed in the Instruction window.

The other four fields in the upper window are shown with inverse video highlighting. The convention followed in the Help facility (and explained in the top-level Help display) is that anything that appears in inverse video is itself a Term for which the user can get Help simply by selecting the highlighted field with the cursor and pressing !!*HELP*!! again. Thus the Terms *HELP* and *SERVICE FACILITIES* are always available whenever the user is getting Help: he has only to move the cursor into either of these fields and press !!*HELP*!!.

The *HELP* Term displays the general information describing the use of the Help system itself, providing the same display as if the user hit !!*HELP*!! with an empty Instruction window.

SERVICE FACILITIES shows an Index-like list of the major topics and instructions in SIGMA: when it is selected, the lower Help window shows a menu of topics on which Help is available. Any Term that appears highlighted (in inverse video) in this list can be selected with the cursor. The user can thus use Help as a menu-driven access facility, or he can type in specific Terms to be accessed (at *New Term*). Whenever the Help display is changed, the *Current Term* field is changed to show the Term whose documentation is being displayed.

The fields *BACK* and *FORWARD* are also shown highlighted. These are not really Terms, but are "virtual function keys" which allow the user to retrace his steps through previous Terms for which Help has been displayed.

2.7.2.2 Requesting help

If there is an instruction or part of an instruction in the Instruction window when the user presses the !!*HELP*!! key, Help is provided for the term corresponding to the instruction word. If the Instruction window is empty, pressing !!*HELP*!! results in a "top-level" Help display which is a description of how to use the Help facility itself.

2.7.2.3 Use of the screen in help

When Help is activated, the Feedback line displays a message telling the user that Help is being shown below and how to "get out" (that is, how to return to what he was doing before he hit !!*HELP*!!). The Alert and Status lines are unaffected. The Instruction window and work area (Display window and/or View window) are mapped away, and the remainder of the screen is divided into two windows: the upper is used to select the documentation to be presented (as described below), while the lower shows the text of the documentation. When the user returns from Help, the screen is returned to the state it had before he hit !!*HELP*!!.

2.7.3 Tutor

The final level of detail in on-line documentation of SIGMA consists of a full curriculum of on-line *Lessons* and *Exercises* covering most of the features of the system. The primary goal of the Tutor is that the user be able to take Lessons on-line and try out various instructions. The Tutor guarantees that the user can do no harm when taking a Lesson.

The Tutor supports two related features: Lessons and Exercises. A Lesson is a detailed description of some aspect of SIGMA. There are a dozen Lessons available. No order is enforced, though the Lessons are arranged in a logical sequence for most users' needs. A user can retake a Lesson any number of times, can quit in the middle, and can start up in the middle the next time.

2.7.3.1 On-line lessons

The user asks for a lesson with the typed instruction **LESSON**, which takes a lesson number as an argument. The available lessons are listed in the hardcopy *SIGMA Reference Manual* [34] and in the on-line Help. When the user executes the **LESSON** instruction, the Lesson text is displayed in the working area of the screen. The Feedback line shows a message telling the user he is in a Lesson and how to get back to what he was doing before he entered the Lesson. The Lesson text shown in the working area can be scrolled, like any display on the SIGMA terminal. Lessons provide complete discussions of the most important topics having to do with using SIGMA. Most Lessons have associated Exercises, and the user is encouraged to use them.

2.7.3.2 On-line exercises

An Exercise is generally a very short and specific task that the user can try in the Tutor's *protected mode*, a special operating mode in which the user is prevented from harming any real data. Lesson 2, for example, discusses a user's special data object called the *Pending File*, and then suggests that the user try Exercise 1 to display a Pending File. Later in the Lesson, the user is shown how to display messages from a file, and Exercise 2, which allows the user to display a message, is suggested.

In keeping with the nonassertive philosophy of the Tutor, the user is not coerced in any way into trying the Exercises. He can skip some or all of them, can take them in any order, and can retake them any number of times.

The user takes an Exercise by using the typed **EXERCISE** instruction, giving the number of the desired Exercise. The Exercise number automatically refers to that Exercise for the Lesson in progress. When the user enters the Exercise, the working area of the screen (which had displayed the Lesson) is remapped to display the Exercise. The Feedback line displays a message telling the user he is in an Exercise and how to get out of it.

An Exercise describes how to specify some particular instruction or set of instructions and suggests that the user try them. In order to try them, the user simply moves the cursor into the Instruction window and types the instruction, just as he would if he were not in the Tutor. At this point, the Instruction window is parsed by SIGMA as always. However, the resultant parsed instruction is not immediately executed; it is first checked to see if it is an allowed instruction for this Exercise. If there is a match, the Tutor allows SIGMA to execute the instruction; otherwise, it displays a message in the Feedback line telling the user the instruction he typed did not match any of those in the Exercise.

When the user executes an instruction within an Exercise, the working area of the screen is presented just as it would be if it were not an Exercise. However, the Feedback line informs the user that he may switch back and forth with the Exercise text by pressing !!*HELP*!! or return to the Lesson by pressing !!*PROMPT*!!.

2.8 EDITING FACILITIES

Before we delve into SIGMA's message service functions, it is best to understand the general text editing facilities provided in the SIGMA system, performed in part in the terminal, which gives rapid response for simple operations (e.g., enter or delete characters via the keyboard, scroll a window). The more complex operations are performed by SIGMA software in the host computer.

Each editing function local to the terminal is associated with a key on the keyboard. The standard typewriter keys cause character insertion at the cursor location (if data entry is allowed at that spot). The inserted character appears where the cursor was, and the rest of the characters on the line move right one position. Overflow from a text line wraps to the line below; it occurs on word boundaries. There is no "overtype" mode provided in the terminal; the user must delete and then enter to replace one string with another. "Overtype" was purposely not provided because it was felt the consistency of a single mode of operation was more important than any performance increase "overtype" would provide.

The !!*RETURN*!! key starts a new line with whatever character is at the cursor position. If the cursor is at the end of the text being edited, it acts like a standard typewriter. The terminal retains the attribute that this text starts on a new line (it is called a *formatted* line). Inserted text on the line above will not overflow onto this formatted line, but will cause a new line to be inserted above the formatted line instead.

The arrow keys (→, ←, ↓, ↑) move the cursor in the direction indicated. However, the system does not permit the cursor to be placed in areas of the screen which have not been specifically defined to be "enterable". Pushing an arrow key causes the cursor to move to the next "enterable" character position, which may be more than one character position away. Text is considered as a concatenated string, so moving the cursor right from the end of a line of text puts the cursor at the beginning of the next line. If there is no enterable character position in the direction requested, the terminal "beeps".

The up (and down) arrows attempt to put the cursor in the character position above (or below) its current location. If there is not an enterable character position there, it will find the closest enterable position to it.

The !!*WORD LEFT*!! and !!*WORD RIGHT*!! keys cause the cursor to move a word at a time. The !!*FWD*!! key moves the cursor to the end of the current line, and subsequent pushes move the cursor to the beginning of the next line, to the end of that line, etc.; !!*BACK*!! acts the same way in the opposite direction.

These cursor movement keys are transparent to window boundaries, and will move into an adjacent window if that is the next proper location. The !!*UP WINDOW*!! and !!*DOWN WINDOW*!! keys jump the cursor into the last known cursor position in the adjacent window. This is usually the most convenient way to move the cursor between the Instruction window and the working window.

The !!*ROLL UP*!! and !!*ROLL DOWN*!! keys cause the data to scroll in the window containing the cursor.

Pushing the !!*DEL*!! key deletes the character at the cursor position. Characters to the right of the cursor move left one position. The !!*DEL*!! key with a left arrow over it deletes the character to the left of the cursor (equivalent to moving left one position, then deleting). Holding the shift key down and pushing !!*WORD LEFT*!! or !!*WORD RIGHT*!! deletes the word to the left or right of the cursor. Similarly, shifted !!*FWD*!! and !!*BACK*!! delete the contents of the line to the left or right of the cursor, respectively.

So long as the cursor remains on the same line, deleting characters merely shortens the line. When the cursor is moved off that line, data from the line below is pulled up to fill out the line. This will not occur if that next line is formatted.

The other "local" editing key is called !!*HERE*!!. This key is used to mark the character at the cursor for some subsequent instruction. Pushing !!*HERE*!! inverts the video presentation of the character to give the user visual feedback, and SIGMA is informed of the character in a manner that is not sensitive to subsequent editing of text around the marked character. The marked character itself is made noneditable. When the next instruction is executed (function key or !!*EXECUTE*!! is pushed), SIGMA will interpret any !!*HERE*!! markers as parameters to the instruction. A part of the instruction execution clears all !!*HERE*!!s.

The terminal editor is not very forgiving if a user puts in a !!*RETURN*!! he later wishes to remove. The deletion of !!*RETURN*!!s is a familiar problem in word processing systems because they are not normally editable symbols. SIGMA provides a function key, !!*UPDATE*!!, which will format the open text (text-object or message) into paragraphs. A paragraph boundary is defined to be a blank line or any formatted line (one which was initiated with a !!*RETURN*!!) which has at least one leading space. This allows the user to enter tables, equations, etc., and keep them formatted as they were entered. !!*UPDATE*!!ing closes up text in the paragraph by pulling out extra blank spaces and removing all formatted line attributes except paragraphs. The user may specify reformatting of only a portion of the open text by bracketing the text to be formatted with !!*HERE*!!s prior to pressing !!*UPDATE*!!.

All other editing operations are performed by executing SIGMA instructions. These operations generally require data access to a larger context than can be held in the terminal.

!!*PICKUP*!!, when initiated with the function key, "picks up," i.e., deletes from the object but holds in temporary storage, the text between two !!*HERE*!!s. Subsequent execution of the !!*PUT*!! function key inserts the data from that temporary buffer at the cursor location, assuming it is an editable location. The !!*MOVE*!! function key is the exact equivalent of doing a !!*PICKUP*!! and a !!*PUT*!!. The !!*COPY*!!

function key is like !!*MOVE*!! except that it copies the data between two !!*HERE*!!s into the temporary buffer without deleting it.

The typed **PICKUP** instruction allows the user to specify the name of a permanent buffer (see 2.15) for storing the text. It also requires two !!*HERE*!!s to identify the text to be picked up. The typed **PUT** instruction allows the specification of any of these named text-objects for insertion into the text on screen at the location indicated with a !!*HERE*!!.

The typed **FIND STRING** instruction causes SIGMA to search the contents of the currently displayed object for the first occurrence of the specified string. The screen will be scrolled (or rewritten) by SIGMA so this occurrence is on screen and the cursor is placed at its beginning.

2.9 DATA OBJECTS

SIGMA deals with four types of objects: *Messages*, *Files*, *Text-Objects*, and *Selectors*. Messages, of course, are the primary data in a message handling system, and they occupy the largest share of the disk file storage in SIGMA. The majority of messages on the system arrive from AUTODIN and are passed to SIGMA by the LDMX.[5] Messages are stored just once in the system, and users share access to them.

SIGMA files, also called *folders* in SIGMA design documents to distinguish them from TENEX disk files, may be thought of as collections of messages. In fact a file does not hold the actual messages, but contains *entries*, which are pointers to messages. An entry is an abstract of the important fields of a message, (e.g., **From**, **Subject**, or **DTG**[6]). When a user displays a file, he can thus recognize the messages it contains. There are ten different types of entries that may appear in a file (see Table 2-1). A user may create and destroy his own files, and he may share access to other users' files. Each user has a special file called *Pending*, which is the delivery point for messages sent to that user.

Selectors are user-created objects that are helpful in selecting certain classes of messages from a file. Their use is described in detail in section 2.14.6, page 2-29.

Text-objects are named entities that contain free text. These can be used for storing address lists, message body paragraphs, routing lists (see section 2.14.8, page 2-30), reports, letters, etc. Text-objects may also be used in conjunction with **PICKUP** and **PUT** instructions.

2.9.1 General Operations on Data Objects

SIGMA allows a user to have *open* (i.e., available for editing) simultaneously one message, one file, and one text-object. An object is opened by the **DISPLAY** instruction, which also causes it to appear in the Display window. Selectors cannot be edited, so SIGMA does not allow the user to **DISPLAY** a selector.

If an object is open, and the user executes a **DISPLAY** of the same type of object, the open object is closed, and any editing changes are made permanent (i.e., the master copy on disk is updated). This updating process

---

[5] Local Digital Message eXchange, the AUTODIN message terminal computer in use at CINCPAC.

[6] Date-Time-Group, the military timestamp for an outgoing message

is called *finishing*; it may also be activated directly by the !!*FINISH*!! function key. A user may close an object without updating it with the **ABORT** instruction. Any editing changes made since opening the object are lost if it is closed with an **ABORT**.

If the user is displaying a file, and then asks to display a message, the message will be put into the Display window but the file will remain open. The user can return the file to the Display window (without closing the message) via the !!*SHOW FILE*!! function key. There are also !!*SHOW MESSAGE*!! and !!*SHOW TEXT*!! keys for switching the other open objects onto the screen. These function keys respond quickly because the terminal retains the display data for each open object.

Messages, selectors, and text-objects may be viewed without opening them by the **VIEW** instruction.[7] A **VIEW**ed object appears in the View window of the terminal. If there is an open object, the View window will occupy the lower 9 lines of the screen. If there is no open object, the View window will occupy the full working space on screen. If nothing is being **VIEW**ed, the Display window occupies that full working space. A **VIEW**ed object is removed by the !!*CLEAR VIEW*!! function key.

Since a **VIEW**ed object is not open, it may not be edited. SIGMA allows the user to put his cursor in the View window, but the terminal editing keys (with the exception of the !!*HERE*!! key) will not function there. A user may copy text out of the View window with the **COPY** instruction, but if he attempts to **PICKUP** or **MOVE** text, SIGMA will not perform the text delete portion of the operation. Thus **MOVE TEXT** gets converted to **COPY TEXT**.

A user can generate a new object with the **CREATE** instruction. The user must specify the type of object, its security classification, and its name as arguments to the instruction. For messages the user must also specify the type of message being created (see section 2.11, page 2-13). **CREATE MESSAGE** and **CREATE TEXT** also open and display the object so the user can type data into it. Creating a file or a selector produces the object but does not open it.

Files, selectors, and text-objects may be deleted from the database with the **DELETE** instruction. A user may delete only his own objects. Messages are created in a shared database where they are considered to belong to the system, and may not be deleted. As a message becomes too old to remain on-line, it is put into a magnetic tape archive. Messages on archive cannot be accessed directly, but can be retrieved from the archive upon request (see section 2.12, page 2-19).

Data objects may be shared by SIGMA users. By design, messages are always shared, although there are certain constraints on access to them (see section 4.12.7, page 4-82). To access a file, selector, or text-object belonging to another SIGMA user, one must perform a **GET** operation, specifying the name and type of the object to be accessed and its owner. An optional argument of **GET** is the name by which this user wants to refer to the object; if this argument is not provided, SIGMA defaults the name to the owner's name. The **GET** will succeed only if the appropriate access control constraints allow it (see section 3.3.2.19, page 3-24).

GETting a file gives the user a pointer to the file, so the user will always have access to the current version. **GET**ting a selector or text-object, on the other hand, creates for the user his own replica of the original object. Changes to the original object will not be reflected in this new object.

---

[7] SIGMA does not permit the user to View a file because of implementation considerations; see 4.7.8.2

One final facility common to data objects is the **DIRECTORY** instruction. Each user has his own *directories* for files, selectors, and text-objects. A *File Directory* is an index to the names of all the SIGMA files that the user can directly access, the classification of each, and its owner. Selector and Text directories are corresponding indexes of selectors and text-objects for a user. There is no message directory; this is the function of SIGMA files. Function keys are provided for Viewing each directory. In addition, there is a typed version of the **VIEW** instruction that allows a user to View other users' directories (assuming access is allowed).

2.10 SECURITY

Before we consider the more specific message and file handling operations in SIGMA, one more general topic should be discussed: Security. The paper "Design of a message processing system for a multilevel secure environment" [3] discusses the SIGMA security model and why it was chosen. Although the system did not implement this security model and made no claim to actually "be" multilevel secure, it did provide a user interface that behaved as though the SIGMA security model had been provided. The effect of multilevel security on the user could thus be observed.

The basic premise of the security model is that one cannot "trust" SIGMA's large amount of code. The only trusted parts of the system are the terminal, the user, the operating system, and a small kernel of security-relevant code called the "trusted job". Whenever a security-relevant instruction is executed, the trusted job must verify that the user really wants to perform that operation. Function keys are interpreted directly by the trusted job and are accepted at face value. However, typed security-relevant instructions must be "acknowledged" by the user. The user executes the instruction in the normal manner, but after the confirmation step the trusted job asks the user to acknowledge the operation by pressing the !!*YES*!! function key. Pressing !!*NO*!! or !!*CANCEL*!! aborts the instruction. The SIGMA security model also inhibits the user from accessing objects at a higher classification than he is allowed and prevents him from putting classified data into an object of lower classification (e.g., filing a Secret message into an Unclassified file).

Since the object data displayed on screen comes from untrusted code, the classification shown in the object itself is suspect. The trusted job controls the terminal's cursor and window security lights to give a positive indication of the classification of each data object. This way the user can verify that the classification shown on screen matches the classification indicated by the terminal security lights.

2.11 MESSAGES

SIGMA supports three different message types: *AUTODIN*, *Memos*, and *Notes*. For each type SIGMA distinguishes between Preparation and Transmitted messages.

2.11.1 AUTODIN Incoming Messages

AUTODIN messages are formal record traffic passed to SIGMA from AUTODIN by the LDMX, or sent out from SIGMA to LDMX (and thence to the addressees via AUTODIN). Figure 2-3 illustrates SIGMA's presentation of an incoming AUTODIN message. AUTODIN message originators (**From** field) and addressees (**To** and **Info** fields) are the commanders of military organizations (e.g., Commander, Naval Electronics Systems Command [COMNAVELEX]). Since SIGMA users are offices and people within such an organization, SIGMA does not use the address fields of an AUTODIN message for its internal delivery, but instead looks at two special internal distribution fields.

```
Ref Id: SEQ 2100585
CONFIDENTIAL [Autodin - Transmitted]  PSN:752348
R(ROUTINE)
221647Z MAY 79
From:   CINCPAC HONOLULU HI
To:      JCS WASH DC
Info:    CINCPAC HONOLULU HI
         NAVSTA SUBIC BAY RP
Orig:   J3
Internal: MME J324
Text:
C O N F I D E N T I A L
SUBJ: FORTHCOMING EXERCISE
A. MY 04001Z JAN 79
THE UPCOMING EXERCISE NEEDS FURTHER DEFINITION.  THE GOALS
SPECIFIED SO FAR DO NOT DETERMINE THE SCOPE OF THE OPERATION.
DECL
NNNN
 ---END OF MESSAGE---
```

Figure 2-3: Incoming AUTODIN message

The usual practice within the community of AUTODIN users is for each command to determine who within the organization should be assigned responsibility for the message and who should receive copies. When they generate outgoing messages, they attempt to guide the recipients' distribution by naming within the body of the message those people who should get copies. There is no standard for this practice, and, since few organizations have the same algorithm for assigning distribution, the results are inconsistent.

At CINCPAC it is the LDMX that scans incoming AUTODIN messages and assigns internal distribution (who will receive copies and who will be responsible for action on the message). This is accomplished by means of a set of preprogrammed criteria. These assignments are appended to the incoming message as two additional fields: **Internal**, for internal distribution, and **Action**, for responsibility. The action assignment, however, is labeled **Action** only for incoming messages where CINCPAC is in the **To** field. If CINCPAC appears in the **Info** field, the action assignment field is called **Cog** (for Cognizance). Back copies[8] of outgoing messages from CINCPAC are assigned to the originating group within the headquarters, and the field is labeled **Orig**.

The receiving LDMX assigns a unique Processing Serial Number (PSN) to each incoming message. This PSN is used internally by the LDMX, and occasionally an Action officer needs to know this number, so it is printed on the message copies and is passed on to SIGMA as an additional field.

SIGMA puts its own sequence number on an incoming AUTODIN message. This is a name which SIGMA uses to uniquely identify the message. This sequence number may be used to specify the message as an argument to an instruction, such as

```
††DISPLAY MESSAGE SEQ 271374
```

[8] Confirmation copies returned from LDMX

although this is not the usual practice. Users would like to be able to refer to messages by the combination of sender and DTG (this is the common way messages are identified in the military community). SIGMA uses this scheme for internal memos, but because of design considerations (see section 4.11.1, page 4-65) it cannot use it for AUTODIN incoming messages.

The text of the AUTODIN message is up to the author, although certain standards are mutually agreed upon by the different branches of the service. The first line of the text contains the classification of the message in clear text. Special handling instructions, Standard Subject Index Codes (SSIC), and/or passing instructions[9] may follow the classification. The subject of the message should appear on its own line after the classification line. References follow the subject; they are lettered in sequential order, and each starts on a new line. Text paragraphs are numbered. Downgrading instructions[10] appear on the final line of the body of the message. The string "NNNN" denotes the end of the message.

Unfortunately, not every organization using AUTODIN follows these standards. Passing instructions come in all forms. Not all messages have subjects. References seldom appear in the standard format. Special markings to indicate AUTODIN Exercise messages seem to conflict with other standards. And, to top it all off, each command seems free to adopt its own variation of format. It is, as a result, very difficult to reliably extract special information from the body of an AUTODIN message (see section 3.2.5.3 for difficulties encountered in subject extraction for building file entries). SIGMA therefore presents the text field just as it comes in and does not highlight the subject or reference fields.

Incoming messages cannot be edited, but users may add comments to them. The user indicates with a !!*HERE*!! where he wants the comment to appear and executes the **COMMENT** instruction. This is a typed instruction because it takes an access control argument. If the access control is specified as *public*, the comment will appear to anyone who reads the message. If the user specifies another user's name with the **COMMENT** instruction (*personal*), only that user and the author of the comment will see it. If no access argument is given, the comment is made *private* to its author.

When **COMMENT** is executed, the text of the message is opened up at the line indicated and an editable comment field appears in inverted video. The user now types in his annotation. When the message is !!*FINISH*!!ed, this comment becomes a part of the message database.

SIGMA also allows the user to "highlight" portions of text in a received message. The user specifies with !!*HERE*!!s the beginning and end of the text to be highlighted and executes the **HIGHLIGHT** instruction. The text is highlighted in inverted video. **HIGHLIGHT** has the same kind of access control attributes as **COMMENT**.

2.11.2 Preparation AUTODIN Messages

Figure 2-4 illustrates a Preparation AUTODIN message just after the user has created it. The top line of the message presents the unique *message identifier* SIGMA assigns to that message (a combination of an internal DTG plus the drafter's name). Users can access this message directly by providing that identifier as the argument to a **DISPLAY** instruction (this is not the usual way to access a message).

---

[9] See NTP3 [30] for discussion of these special markings

[10] See NTP3 [30]

```
Ref Id: J324 220018Z MAY 79 (J324's Version)
CONFIDENTIAL [Autodin - Preparation]
Briefing Memo:
        Precedence: R(ROUTINE)
                From:   CINCPAC HONOLULU HI
                To:
                Info:
                Exempt:

                C O N F I D E N T I A L
        Subj:
        Ref:
        Text:
        Downgrade Instructions: DECL
Chop:
Release:
Originating office: J3
Distribution: J324
 ---END OF MESSAGE---
```

Figure 2-4: Preparation of AUTODIN message (before Chop)

The **Briefing Memo** is a public comment at the top of the Preparation message where the drafter can put information to be used during coordination (see the description of coordination, section 2.13.2, page 2-21). This field is stripped before the message goes to the LDMX.

The **Precedence** field is restricted to one of five possible values (Routine, Priority, Immediate, Flash, Emergency). It is automatically filled to be Routine, but is left editable so the user can change it. He needs only to change the first letter in the field, as this is what SIGMA uses when it parses the field. The user may also add a second precedence after the first one if he wants Info addresses to have a lower precedence.

The **From** field, like **Precedence**, is automatically filled in to be CINCPAC. The field is left editable in case the user wishes to send the message from some other organization. SIGMA does not parse this field but simply passes it on to LDMX. The user fills in the address fields (**To**, **Info**, **Exempt**[11]), putting one addressee per line. The !!*RETURN*!! key will open space for another addressee. SIGMA does not have access to the Plain Language Address Tables (PLAD) in LDMX, so it cannot verify these addressees.

The classification of a Preparation message is specified at the time the message is created, so the classification is not editable. The user may add data at the end of the **Classification** line, however. This is normally where the *Standard Subject Indicator Code* (SSIC), special handling, and passing instructions go. If an unclassified message is to be handled as *Encrypt For Transmission Only* (EFTO), the user types "EFTO" on this line immediately following the UNCLASS.

One major difference between the DD173 (military optical character reader message preparation form) and the SIGMA Preparation message form is that SIGMA has a specific **Subject** field where the user is expected to enter the message subject. Upon release, SIGMA will put that subject (along with the herald "SUBJ:")

[11] See NTP3 [30] for usage of the Exempt field

into the outgoing message body at the proper place (after the **Classification** line). This guarantees that the back copy coming to SIGMA from LDMX will be correctly parsed. Similarly, SIGMA provides a distinct **Ref** field so that references can be parsed. However, parsing references is one of the features that was never implemented in SIGMA (see section 3.3.2.12, page 3-17).

The **Text** field expands as necessary. The amount of text that may be put in is essentially unlimited. The AUTODIN Preparation message form indents the **Text** field 16 spaces so the longest line that can be passed to the LDMX is 63 characters, 6 characters under the limit of 69 that LDMX will accept.

SIGMA has the facility to reformat text, so the user does not need to worry about ragged lines left around from editing. The terminal does part of this, since it automatically wraps on word boundaries and closes up lines when text is deleted. However, if the user does a carriage return in the text, the terminal will treat that as the beginning of a new *formatted* line and will honor that forever unless SIGMA changes it. The only way for a user to remove the carriage return entered from the keyboard is to ask SIGMA to reformat the text, via the *!!UPDATE!!* function key (see section 2.8, page 2-9).

Preparation messages may be commented, but the comments appear at the end of the paragraph indicated by the *!!HERE!!* rather than at the end of the line. Highlights are not allowed in Preparation messages.

The **Downgrade Instruction** field appears only in classified messages and is preloaded as "DECL."[12] It is editable, so the user can specify a date or can change the instruction. When the message is released, the herald saying **Downgrade Instruction** is removed, and only the contents of the field are put into the message.

The last four lines of the AUTODIN Preparation message are fields to allow coordination and to define distribution assignment for back copies. The **Chop** field contains the names of users the drafter wants to "chop" (military jargon for approve) the message before release. The **Release** field specifies his choice for the releaser of the message (section 2.13.2, page 2-21 describes the use of these fields in more detail). The **Originating office** is the name of the Directorate from which the message will be released. SIGMA fills this in with J3, since that is the directorate primarily served by the MME, but the user may specify another directorate. The **Distribution** field specifies who should receive back copies when this message is released.

2.11.3 Memos

Memos are used to transmit formal communications within CINCPAC headquarters. At the users' request, the format of Memos in SIGMA was made to look as much like CINCPAC's standard paper memos as possible. Like AUTODIN messages, memos come in two forms, *Preparation* and *Transmitted*.

Figure 2-5 shows a Preparation Memo. The **Briefing Memo**, **Chop**, and **Release** fields serve the same role as they do in an AUTODIN Preparation message. The **Date** field is automatically filled in with the date the memo is released. Note that memos do not have precedence. The **From** field is preloaded by SIGMA with the drafter's log-in name, but it may be edited so that a user can prepare a memo from another person. The **To** field and the **Copy to** fields are interpreted by SIGMA when the memo is released. If SIGMA does not recognize a name in either of these fields to be a SIGMA user, it will put an asterisk after the name to indicate it cannot deliver the memo to this user.

---

[12] See NTP3 [30] for usage of Downgrade instructions

```
        Directorate/Memo/Memo Number:   821
                                Date:   21 MAY 79

SECRET

MEMORANDUM

From:   J324
To:     J3

Subj:   Test of Coordination of memos

Ref:

Encl:

Text:
Memos, as well as Autodin messages, may be coordinated using SIGMA.

              Signature block:J31
                              Lt. Abel Cain USN
                              J324

                                      SECRET

Copy to:
DDO

Orig:   J324
Typed by:JBW

Briefing Memo: I want your advice on who else should get this memo.
Chop:   J31   J32   J322
Release:
Ref Id: J324 212344Z MAY 79 (J324's Version)
SECRET [Memorandum - Preparation]
 ---END OF MESSAGE---
```

Figure 2-5: Preparation Memo (Chopped)

The **Encl**, **Orig**, **Typed by**, and **Signature block** fields are fields of a standard CINCPAC memo, uninterpreted by SIGMA. The **Ref** field appears at the bottom of a memo. Its role is a unique identifier for the memo, just as for an AUTODIN Preparation message.

The Transmitted Memo is generated by SIGMA from the Preparation Memo when it is released. The fields of a Transmitted memo are not editable, but the user may add comments. Copies of a released memo are delivered to the memo's addressees. In addition, back copies of the memo are supplied for the Releaser and anyone he puts into his **Chop** or **Release** field. To satisfy CINCPAC's desire for administrative control, SIGMA also delivers a back copy of all released memos to J3's Administration Office, J301.

2.11.4 Notes

Notes are intended for local informal communication and, as illustrated in Figure 2-6, have a simple format. Notes cannot be coordinated, but Preparation Notes are still edited and then released to become Transmitted Notes. At one time the plan was to underscore the informality of Notes by not archiving them, so that after some period of time they would just vanish. The users did not like this idea, so Notes are archived like all other messages.

```
Ref Id: J324 220043Z MAY 79
CONFIDENTIAL [Note - Transmitted]
From:    J324
To:      J301
CC:
Subj:    Meeting tomorrow
Ref:
Text:
Can we move the meeting on SIGMA tomorrow from morning to afternoon?
Releaser: J324
 ---END OF MESSAGE---
```

Figure 2-6: A SIGMA note

2.12 INCOMING MESSAGE OPERATIONS

The most fundamental operation in a message system is to **DISPLAY** a message. As mentioned earlier, this opens the message so it may be edited or commented on. The user may prefer to **VIEW** the message, if he does not expect to edit it. The user may specify the message to be displayed or viewed by referring to it from the open file (see section 2.14, page 2-24) or by typing in the **Ref** of the message (either sequence number for AUTODIN received, or Message Identifier for Memos, Notes, and Preparation AUTODIN).

If the user tries to access (**DISPLAY** or **VIEW**) an old message, he will often be told

```
**Acknowledge YES to request the
retrieval of <message>
```

Old messages (messages that have been on-line 30 days without being referenced) are moved to a secondary tape storage archive. To get the message retrieved the user pushes !!*YES*!!, which sends a special request to the operator on the computer room floor. After the operator has gotten the proper tape and loaded the message back into the on-line database, a *Retrieved* entry appears in the user's Pending file. This operation normally takes 5 to 10 minutes. If the user prefers not to bother with the message, he may respond !!*NO*!! when he is asked to acknowledge the retrieval.

Incoming messages may be forwarded to other users in two ways:

1. The **FORWARD** instruction sends the recipient the message for his information (*For_Info* entry), and appends the recipient's name to the **Internal** field.

2. The **ACTION** instruction sends the recipient the message for his action (*For_Action* entry), and appends the recipient's name to the **Action** field. In this case, the name of the user assigning action is also included, resulting in an **Action** field looking something like the following:

```
Action: J3 J31(by J301) J312(by J31)
```

   This allows others to follow the chain of action assignments.

The **ACTION** instruction also puts a *File_Copy* entry (see 2.14.1) into the the action assigner's file bearing the special name *ACTION_LOG*. This may be the user's own file, or one he has gotten from some other user. The intent of this file is to allow an organization to track the action on the message from a central place. It is especially designed to support crisis action teams. If the rest of the team shares access to this action log, they can record their action status by writing comments on the entries.

A message may also be filed into any files that the user can access. The **FILE** instruction puts an entry into the named file. There is no limit to the number of files into which the message can be put. It may also be filed into the same file several times.

A limitation of **FORWARD**, **ACTION**, and **FILE** instructions is that they may be applied only to a single message and to a single recipient at a time. See the discussion of the **ROUTE** instruction for a more powerful operation (section 2.14.8, page 2-30).

2.13 OUTGOING MESSAGE OPERATIONS

2.13.1 Preparing the Draft

Messages may be created in several different ways, the most straightforward of which is to execute a **CREATE MESSAGE** instruction, specifying the type of message (AUTODIN, Memo, Note) and its classification.

If the message is a reply to an Incoming message, the user may prefer to use the **REPLY** instruction. This creates a preparation message of the same type as the message it is answering (the original). In addition, it automatically fills in the addressees for the message from the original. The **To** field is copied from the **From** field in the original. The **Info** field (**Copy to** for Memos, **CC** for Notes) is filled in with the names of all the **Info**, **Copy to**, or **CC** addressees from the original. The address fields are editable, so they may be easily modified.

The !!*REPLY ENTRY*!! and !!*REPLY NEXT*!! function keys allow the user to reply to the Current, Next, or an indicated (with a !!*HERE*!!) message from a file. Sometimes a **REPLY** should not carry the same classification as the original message. In this case the user may type

```
††REPLY MESSAGE
<Classification>
```

into the instruction window, and specify the classification he wants for the **REPLY**.

Often a user sends a standard message out on a regular basis, with only minor variations. SIGMA eases the job of producing these messages by allowing a user to **COPY** an already existing Preparation message. This message may be filled in as much as desired. SIGMA creates a new message and copies into it all the fields from the specified original. This can then be edited into final form.

A special form of outgoing request is Readdressal of an incoming AUTODIN message (i.e., sending it on to a specified addressee, imbedded within a new AUTODIN message). LDMX already supports producing such a Readdressal message, but it requires a special form of message from SIGMA to trigger it. This message form looks like a standard AUTODIN header, but the body contains just the word "RADDR," followed by the PSN of the message, and on the next line the DTG and sender of the readdressed message. In SIGMA the user specifies the message to be readdressed and executes the **READDRESS** instruction. A standard SIGMA preparation AUTODIN message is generated with the special Readdressal request format automatically filled in. This data is editable, so the user may alter its contents. Except for its special form, a Readdressal request message is treated like any other AUTODIN Preparation message in SIGMA.

2.13.2 Coordination

SIGMA is designed to assist in the process of reviewing AUTODIN messages and Memos preparatory to their being released. This *coordination* process is essentially a message service within a message service. That is, the drafter creates the initial version of the message and sends it to the people he wishes to review it. Only those people who are sent the message may read it. These reviewers (called *coordinators*) read and possibly edit their versions of the draft, then indicate their overall disposition ("chop"). The drafter, after reading the comments and changes from the first round of coordination, makes whatever alterations he feels are needed and sends the modified draft out again to the same or to other reviewers. The process may take as many iterations as are necessary. When the message is ready for release, it is sent to the person designated *Releaser*. This section describes coordination in SIGMA for AUTODIN messages in more detail. The same process applies to Memos. Notes cannot be coordinated.

As Figure 2-4 illustrates, there are a number of fields in a Preparation AUTODIN message that do not appear in the message that eventually goes out to the addressees (the transmitted message). These same fields appear in Preparation Memos as well. These fields are used in the Coordination phase and are stripped off at the time of release.

The **Briefing Memo** is a text field where the drafter normally explains background information or special instructions about the message. It appears before the draft message because it is information that generally should be read prior to seeing the message.

The **Chop** field appears after the body of the draft message. This is where the drafter specifies the users he wishes to review the message. The **Release** field contains the name of the drafter's intended releaser. The reviewers who are sent this draft are able to see the drafter's **Chop** and **Release** fields.

The field labeled **Originating office** is required by LDMX. It contains the office code of the Directorate from which the message is emanating. Since MME is primarily serving J[illegible] SIGMA fills in J3 automatically at the time the message is created. It may be edited.

The **Distribution** field specifies who will get back copies. At one time SIGMA automatically appended the names of each coordinator, the releaser, and the drafter to this field on the theory that they were all interested in seeing that the message got out. However, LDMX at CINCPAC does not normally distribute to office codes below the level of the Directorate (e.g., J3), and the Communications Center personnel objected to having to handle the paper copies produced for these additional offices. A compromise was struck, and SIGMA was changed to append only the office code of the releaser to the **Distribution** field.

When the drafter has filled in the message and briefing memo completely, he enters into the **Chop** and **Release** fields all the office codes he plans to coordinate with. He next must decide which people on the **Chop** list he wants to review the message first. He indicates these by marking their names in the **Chop** list with !!*HERE*!!s and presses the !!*COORDINATE*!! function key. SIGMA responds by highlighting the names selected that it recognizes as SIGMA users and indicating any unrecognized user names (i.e., not SIGMA users) with an asterisk. It is important that the message service be able to accept user names that it cannot recognize and indicate this on the display so the drafter can send these people a hard copy. The drafter is asked to *acknowledge* the coordination to the highlighted users by pushing the !!*YES*!! function key. !!*NO*!! or !!*CANCEL*!! causes the operation to be aborted.

Coordinating a message to a user causes a *For_Chop* entry to be put into his Pending file. This indicates to the coordinator that his chop is requested. When the coordinator displays the message he is given his own copy of the drafter's original. The coordinator may edit this version and/or annotate. He normally would then chop the message, either Yes or No (!!*CHOP YES*!! or !!*CHOP NO*!! function keys). Chopping the message puts a *Chopped* entry into the drafter's Pending file.

Figure 2-7 shows how the Preparation message looks to a coordinator. Note that it shows the drafter's **Chop** and **Release** fields as nonenterable status fields and that new, empty **Chop** and **Release** fields are provided for the coordinator. If the coordinator wishes, he may start his own coordination cycle by typing into his **Chop** or **Release** fields the names of users, then coordinating the message to them in exactly the same way the drafter did. Each *subcoordinator* will get an editable copy of his *delegator*'s (higher-level coordinator) version of the message (not the original draft). The sub-coordinator will see the status of both the coordinator above him and the drafter, and will have **Chop** and **Release** fields of his own. When a subcoordinator chops the message, a *Chopped* entry is delivered to the Pending file of the coordinator that sent the message to him.

Each coordinator (including the drafter and subcoordinators) sees the status of his own coordination and the coordination of anyone above him in the process. Possible states that apply for each name are *Not Sent*, *Not Read*, *Read*, *Chopped Yes*, and *Chopped No*. In addition, if a coordinator has made any changes to the message at all (an edit to a field of the message, a comment, or a name put in the **Chop** or **Release** field) the status shows the letters *ED* (for edited). If the drafter has edited the message since a user has chopped, that user's status shows as *OLD*.

Whenever a user **DISPLAY**s a Preparation message (regardless of what file entry he uses to access it), he always sees his own version. Any comments he makes appear only on his version. In order to see the changes or comments of other users who have been coordinated on the message, he must execute the typed instruction **VIEW VERSION**, specifying the name of the user whose version he wishes to see. This will put into his View window that user's version, where he can compare it with his own and see that user's comments.

A typical scenario in the coordination of messages would have the drafter create the message, fill in the **Chop** and **Release** fields with all the office codes he plans to coordinate with, and send it to a subset of these for the first round. These coordinators would review the message, possibly edit it, and either chop it or coordinate it to someone of their own choosing (someone whose opinion they want before they make a

```
Ref Id: J324 220018Z MAY 79 (J301's Version)
CONFIDENTIAL [Autodin - Preparation]
Briefing Memo: This field will not be included
               in the final released message.
        Precedence: R(ROUTINE)
                From:   CINCPAC HONOLULU HI
                To:     JCS WASH DC
                Info:   CINCPAC HONOLULU HI
                        NAVSTA SUBIC BAY RP
                Exempt:

                C O N F I D E N T I A L
        Subj:   FORTHCOMING EXERCISE
        Ref:    A. MY 04001Z JAN 79
        Text:   THE UPCOMING EXERCISE NEEDS FURTHER DEFINITION.
                THE GOALS SPECIFIED SO FAR DO NOT DETERMINE THE
                SCOPE OF THE OPERATION.
        Downgrade Instructions: DECL
Chop:
Release:
Originating office: J3
Distribution: J324
J324's Chop:  J301:chop_YES by SMITH, J38:not_sent, J31:not_read,
        DDO:not_sent
J324's Release:  J3:not_sent
 ---END OF MESSAGE---
```

Figure 2-7: Coordinator's view of Preparation AUTODIN message (Chopped)

disposition of the message). The drafter monitors the progress by the *Chopped* entries he receives and by looking at the chop status area of the message. If the status shows a particular user has edited his version of the message, the drafter would look at that version. When the drafter has sufficient response to his draft, he edits his version (the initial draft) incorporating whatever changes he desires from the versions or comments of the coordinators. He then starts the second round of coordination by indicating with !!*HERE*!!s the users he wishes to see this new rendition and pressing !!*COORDINATE*!!. Note that he can !!*COORDINATE*!! to anyone on his **Chop** list, including users who received the message in the first round. This causes this new draft version to be substituted for that coordinator's version, so that when he **DISPLAY**s the message he will see the new draft and not his old version. At this point the coordinator can still access his old version by executing a **VIEW VERSION** and indicating his own Log On name as the user. The old version appears in the View window and is not editable. This old version is lost whenever he does a !!*CHOP YES*!!, !!*CHOP NO*!!, or !!*FINISH*!! of the new draft (to preserve the old version he must **ABORT** the new draft message after he has read it).

At any time throughout the coordination process, the drafter or a coordinator may coordinate the message to the user indicated in his **Release** field. This puts a *For_Release* entry in that user's Pending file. This user has all of the options that any other coordinator has, i.e., he may edit or comment on his version and then chop it (Yes or No) back to whoever sent it to him, coordinate it on to someone else (either for chop or for release), or !!*RELEASE*!! it. A user does not have to be in the release field in order to release it. The only limitation on releasing is that the user be authorized to do so. The SSO maintains the database which specifies who may release AUTODIN messages, Memos, and Notes.

2.13.3 Release

When a user does release the message, it is his version that is copied into the Transmitted message. All comments and Preparation-only fields are stripped off. If it is an AUTODIN message, the contents are reformatted for the LDMX and sent. If it is a Memo or Note, a new SIGMA Transmitted message is generated and delivered to those SIGMA users whose names appear in the **To** and **Copy to** (or *CC*) fields.

AUTODIN back copies are sent to the users whose names appear in the **Distribution** field of the released version of the message, and to the Releaser. *Back_Copy* entries of Memos are put into the Pending files of the Releaser, and all of the coordinators. At CINCPAC the J3 Administrative office (J301) controls the issuance of memos, so SIGMA sends a back copy of all Memos to this office as well.

Upon release the Preparation message status is updated to show that it has been released, and further coordination on it is frozen. This Preparation message remains in the database in case someone wishes to look at its coordination history. All of the coordinators' versions, comments, and status remain intact. The released Preparation message is still accessible only by coordinators, the drafter, and the releaser of the message. *Back_Copy* entries point to the transmitted message, but they also show the message identifier of the Preparation message that produced it (see Figure 2-8). This linkage is necessary to allow users to locate the Preparation message containing the coordination history for the outgoing message.

2.14 FILES

Files are the basic message management tool in SIGMA. Although the user may think of files as containing messages, in fact they contain an index to messages in the form of *entries*. A file entry is an abstract of data from a message, plus a pointer to that message in the database. Users access messages through file entries. A user may create and delete his own files, and he may access other users' files. A file is named and given a security level at the time it is created. A file name may be any arbitrary single-word alphabetic or numeric string, except that it must start with an alphabetic. The underline character is treated as an alphabetic, so multiword strings connected with underlines may be used for a file name, e.g.,

```
Merchant_Ship_Sightings_January_79
```

The classification of a file determines the maximum security level of its contents. SIGMA will not allow a user to file a Secret message in an Unclassified file, but an Unclassified message may be filed in a Secret file. Furthermore, a user who logs on as Unclassified will not be able to open a Secret file, even though it may contain nothing but unclassified messages.

2.14.1 File Formats

Files are made up of entries. Each entry is an abstract of a message. Figure 2-8 illustrates the format of a SIGMA file presented to a user. The first piece of information in an entry is its entry number, which serves as a convenient handle for the user to specify arguments to instructions. Entry numbers are sequential and remain fixed while a file is open; entries will be renumbered the next time the file is opened. File entry numbers are shown in inverted video to make them stand out.

Files are not editable. Entries are appended to files through user instructions such as **FILE**, **ACTION**, or **FORWARD**. Entries are removed by executing the **DELETE ENTRY** instruction. A file entry may be annotated via the **COMMENT** instruction in the same way a message may be annotated (see section 2.11.1.

```
File: PENDING Security: SSSS Length: 15
 1 O UU Auto 042222Z DEC 77 From: JCS WASHINGTON DC
        Incoming   Cog: J3
        Subject: AIRCRAFT HIJACKING
 2 R UU Auto 010024Z DEC 77 From: CINCPAC REP PHILIPPINES SUBIC BAY RP
        Incoming   Act: J03
        Subject: INFORMATION ON GAO ACTIVITY
 3   SS Memo J322 020011Z DEC 77 From: J322
        Incoming
        Subject: Check out a message for me.
 4   UU Note AMES 020333Z MAR 79 From: AMES
        Incoming   Subject: Action on Notes
 5 R UU Auto J322 280038Z JAN 79
        for_Chop   By: J324
        Subject: Exercise Procedures
 6 R UU Auto J322 280040Z JAN 79
        for_Releas By: J324
        Subject: Elegant Eagle
 7 R SS Auto 082359Z OCT 76 From: NISC WASHINGTON DC
        for_Info   By: J322 Act: J2 J3(by J322) J324(by J322)
        Subject:  NOFORN/WNINTEL //NO3822// SECTION 01 OF 02
 8 R SS Auto J31 082358Z OCT 78
        File_copy  By: J324
        Subject: RADDR FBIS BANGKOK//120852Z DEC 77
 9   SS Memo J324 070027Z MAR 79 CID: J324 070020Z MAR 79
        Back_copy
        Subject: Handling of last week's exercise
10 R UU Auto 040130Z DEC 77 From: FBIS HONG KONG
        for_Action Act: J2 J324(by J301)
        Subject:  H BBC RUMJ
11 R UU Auto 041647Z DEC 77 From: FBIS OKINAWA JA
        for_Action Act: J2 J301(by J301)
        Subject:  BBC
12 P UU Auto 040555Z DEC 77 From: FBIS BANGKOK
        Retrieved  Act: J2
        Subject:  BBC RUMT
13   SS      AAA 230648Z MAY 79
        ERROR      XMIT-FAIL: Your message had no TO list.
14 R SS Auto J324 232332Z MAY 79
        Chop(Y)-Ed By: J301
        Subject: Exercise Procedures
15   CC Memo J324 232338Z MAY 79
        Chop(N)    By: J301
        Subject: Review Meeting
 ---END OF FILE---
```

Figure 2-8: A Pending file

page 2-15). File entries may also be highlighted. The subject line of a highlighted entry is shown in inverted video.

Since files cannot be edited, SIGMA restricts the cursor from entering the text of the file. This means the cursor will jump from one entry number to the next with a single push of the ↑ key. All instructions that accept a file entry number as an argument will accept a !!*HERE*!! marker on the entry number as well.

There are a variety of types of file entries, depending on what action caused the entry to be made, but their format is basically the same. Next to the entry number is a single-letter precedence indicator (*AUTODIN* only, Memo and Notes show a blank). Next on the first line is a two-letter classification designator. Following that is the message type (AUTO, MEMO, NOTE), then the DTG and From fields (which for all but incoming AUTODIN is also the Ref ID of the message). The second line of the entry is indented for clarity. The first information on this line is the type of entry. Table 2-1 describes the meaning of each entry type. The rest of line 2 and line 3 are dependent on the entry and message type. The subject line appears on line 2 for Notes but gets line 3 to itself for AUTODIN messages and Memos. Most Transmitted Memos and AUTODIN entries show the Action field of the message. Preparation messages usually show who sent it to the user (e.g., "By: J31"). Error entries indicate the type of error that occurred.

Table 2-1: Types of file entries

Incoming:	a message that has been sent to the user.
Error:	an error condition has been detected. A brief description of the error is provided in the entry.
For_Action:	a message sent to this user for his action with ACTION.
For_Info:	a message sent to this user for his info with FORWARD.
For_Chop:	a message sent to this user for his chop with COORDINATE.
For_Releas:	a message sent to this user for his release with COORDINATE.
Chopped:	a message returned as either Chopped(Y) or Chopped(N).
File_Copy:	an entry for a message that has been placed in the file.
Back_Copy:	a comeback copy of a released message.
Retrieved:	a message retrieved from Archive.

The last entry that the user has referenced is called the *Current entry*. It is distinguished by having its number shown as white-on-black rather than inverse video. A number of function keys refer to the Current entry or the *Next entry* (the one immediately following the Current entry). The user may step sequentially through a file of messages by pushing !!*DISPLAY NEXT*!!.

Entries normally appear in files in the order they are received, most recent last. However, SIGMA does provide a SORT instruction which will sort the named file into DTG order, most recent DTG last. This was requested by users because they are used to seeing messages filed that way in the paper system. Since SIGMA provides sophisticated retrieval features (see section 2.14.6, page 2-29), having messages in DTG order does not particularly aid retrieval. However, it has proven to be helpful to users who deal with a paper copy of the file contents. The Readboard is also sorted by Date Time Group, because "that is the way it is done".

For each file in a user's file directory SIGMA keeps a *high water mark*, the highest numbered entry which he has displayed. When he displays the file, this high water mark will be at his Current entry. SIGMA also

remembers which is the last entry when the user !!*FINISH*!!es a file. When he next opens the file he can find out which entries have arrived since the last time he saw the file through the selector *Recent* (see section 2.14.6, page 2-29).

Two function keys are provided to facilitate moving around in a file. !!*GO TO NEXT*!! moves the current entry to the next entry in the subset displayed. Pushing !!*CURRENT ENTRY*!! causes SIGMA to scroll the file to put the Current entry on screen. If an entry is marked with a !!*HERE*!! when !!*CURRENT ENTRY*!! is pressed, that entry will be made the Current entry.

2.14.2 Pending File

Each user has a special file called *Pending*, which is established when he is given a SIGMA account and which he cannot delete. The Pending file is the delivery point for messages sent to that user. It is the analog of his mailbox or in-basket. From the point of view of its manipulation, the Pending file is like any other file. The user may file messages into it, delete messages from it, etc.

There is a Pending file for each office code and user name. When a user logs on as an office, his Pending file is the one for his office. His personal Pending file is also available to him under the special name "MyPending". Thus if Smith logs on as J34, he accesses J34's Pending file as "Pending" and Smith's Pending as "MyPending".

Although a user has control over files he creates, they are maintained by a central process in SIGMA (see section 4.12.5, page 4-79). They may be accessed and updated by other users even though the owner may not be logged on at the time. When a user DISPLAYs a file, he gets a copy of the master file. Any changes he makes during his session (delete entries, comments, etc.) are made immediately to his working copy, but will not appear in the master copy until the user does a !!*FINISH*!! (or DISPLAYs a different file, which does a !!*FINISH*!! for him). For most files the user will not see new entries that are made while he has it open. The one exception is the Pending file. If new messages are sent to a user while he has his Pending file open, they will be delivered to his open working copy as well as to the master copy. This feature applies only to messages sent to the user, not messages FILEd into his Pending file.

2.14.3 Date Files

Incoming AUTODIN message delivery is determined by the Action and Internal Distribution assignments made by LDMX. In addition, SIGMA delivers a copy (really another file entry) to a special file called the *Date File* based on the DTG of the message. There is a Date File for each day that SIGMA has accepted traffic from LDMX.

If SIGMA receives a message with a DTG that does not have a corresponding Date File, a Date File is created and the message is filed into it. Thus SIGMA keeps building new Date Files. These files remain on the system and can be accessed at any time, even though the messages may have long since been archived. The Date Files serve both as a log of the traffic SIGMA has received from LDMX and as the primary database for message retrieval. At the completion of MME there were Date Files on-line for all traffic received by SIGMA during the conduct of the experiment (nearly two years).

MME users found it valuable to search the most current Date File for traffic of interest rather than wait for messages to be routed to them by J301. This way the user can get his messages sooner and, more important, he is able to pick up messages that J301 misrouted which otherwise would escape his attention (see section 2.14.6, page 2-29 for a description of the aids SIGMA provides for searching through a file).

An AUTODIN readdressed message consists of a new message header with its own **From** and **DTG** wrapped around the original AUTODIN message. Similarly, a readdressal of a readdressal will have three **Froms** and **DTGs**. To insure that it can be found by any pertinent reference, SIGMA files incoming readdressed messages into Date Files by all incorporated DTGs.

2.14.4 Readboards

Readboards are collections of messages deemed to be of interest to a particular person or group. Typically a readboad is built for J3, and another for his Executive Officer, J30. Other readboards may be built to be passed around among a number of Action offices. This is a major way message information is disseminated at CINCPAC. Of particular interest are the Readboards given to J3. At CINCPAC this Readboard is built by the watch team in the command center and screened by the Executive Officer. In the paper system J3 gets his own Readboard and other officers can only surmise what is on it. However, the shared nature of files in SIGMA allows other officers to access J3's Readboard and therefore know what messages he will be aware of. This is deemed to be a valuable asset among SIGMA users at CINCPAC.

2.14.5 Deleting Entries

The **DELETE ENTRY** instruction marks a file entry in the open file for deletion and removes it from the user's screen. A user may delete entries only from files he has created. He is also limited to deleting entries that are in the set of entries being displayed (see 2.14.6). Function keys are provided for deleting the Current entry or the Next entry. A user may delete more than one entry by typing the **DELETE ENTRY** instruction with the entry numbers of the messages to be removed. A dash between entry numbers refers to all entries inclusive between those entries. If no entry number is given in front of the dash, SIGMA defaults that value to the first entry of the displayed set. If no entry number is given after the dash, SIGMA defaults to the last entry of the displayed set. Thus the typed instruction

```
††DELETE ENTRY -
```

will delete all the entries currently displayed.

As previously stated, **DELETE ENTRY** only *marks* the entry for deletion. The entries are not actually removed until the file is !!*FINISH*!!ed. Prior to the !!*FINISH*!! the user may **RESTORE** any deleted entry, which removes the deleted mark and restores the entry on screen. The user may also **ABORT** the file, which will throw away all changes he has made to the file (deletes, comments, and highlights) and leave the file in the state it was in before he opened it.

As an aid to users who prepare Readboards, an **EMPTY** instruction was added to SIGMA. This deletes all entries in the named file without opening the file. Although a convenient way to expunge a file of its entries, it is dangerous because such entries cannot be restored.

It has been mentioned that a user may **FILE** the message he is looking at into a named file, which puts a *File_Copy* entry into the file. He may also **FILE** an entry from an open file to a named file, which puts a copy of the entry (without changing its type) into the file. The **MOVE** instruction is the same as a **FILE** and **DELETE** on the entry.

2.14.6 Selection from Files

Perhaps the most useful feature of the SIGMA system is its facility for assisting the user in finding messages in the database. SIGMA allows the user to search a file for all messages meeting a specified criterion. These *selection criteria*, which can be saved as objects called *selectors*,[13] are combinations of attributes of the entries in the file. Table 2-2 lists the attributes that can be used.

The **RESTRICT** instruction takes a selector as an argument and applies it to the open file. It then displays on the user's screen the subset of entries which meet the criteria. A subsequent **RESTRICT** applies to these entries, further restricting them to those that meet both the first and second selection. This is equivalent to the logical AND of the two selectors which, of course, the user could have specified the first time. However, very often it is not obvious that the second selector should be applied until the results of the first **RESTRICT** are seen.

The user can perform the equivalent of ORing two selectors by executing a **RESTRICT** with the first selector, followed by an **AUGMENT** instruction with the second. The user can apply any reasonable number of **RESTRICT**s and **AUGMENT**s to a file.

Predicting the results of applying complex selectors is often difficult, so SIGMA allows the user to take it a step at a time. If the user makes a wrong choice by applying a particular selector, he may use the !!*BACKUP ONE*!! function key to step back in the chain of selections, returning him to the set he had previously. This may be applied sequentially to get back to any previous set of entries or until he backs up all the way to the original file. To return to the original file in one step, **BACKUP ALL** is executed.

Often a user wants to preserve a complex selection so that it can be used over again, without having to type in all of its components. When he has incrementally built up such a selector he executes a **CREATE SELECTOR** instruction, specifying a name he wishes to remember it by. Since this selector may contain strings of text (e.g., contents of subject field) which may be classified, the selector is defined to be at the security level of the open file. To apply this selector at a later time he merely types:

```
††RESTRICT WITH <selector name>
```

If the user prefers, he may define the contents of a selector when he creates it. He does not need to even have a file open to create a selector in this manner. However, in this case, he must specify the security level of the selector. Thus,

```
††CREATE SELECTOR URGENT_JCS UU FROM "JCS" AND NOT
ROUTINE
```

builds an unclassified selector named URGENT_JCS which will select all high-precedence messages from JCS. Another user may do a **GET** on URGENT_JCS, assuming he has access (see 4.3.6), and give it the same or a new name for his own use. This puts a copy of the selector into his database. It will not change if the owner alters the original. SIGMA maintains a directory of selectors for each user, which can be **VIEW**ed.

Typical usage of selection from files occurs when a message references an earlier message by its DTG. To

---

[13] Selection criteria are arbitrary Boolean combinations of attribute values; selectors are named SIGMA objects which can store these selection criteria. Both of these forms can be used in the instructions which perform selection.

see that message in SIGMA, the user **DISPLAY**s the corresponding Date File, then **RESTRICT**s with the DTG of the message. At other times a user will have only a rough idea of the date a desired message came in, but he will know some other attributes of the message, such as whom it was from, its subject , etc. In this case the user works through a series of Date Files, first **DISPLAY**ing the file, then **RESTRICT**ing with the appropriate selector, until he finds the message. This is somewhat tedious, and the users have asked for a single instruction that allows them to search multiple files with the same selector. Even with this limitation, the fact that SIGMA has Date Files and message archives that go back nearly a year and a half means that users have a way of retrieving old messages that they have never had before. The key consideration here is that the user does not have to recognize at the time the message comes in that he may need the message later.

2.14.7 Keywords

One of the attributes on which a user can select (see Table 2-2) is keywords. Keywords must first be assigned by the owner of the file before they can be used in a selector. This is done with the **KEYWORD** instruction, which takes as arguments an entry and the keyword to be assigned. The same entry may have more than one keyword, and a keyword may be assigned to more than one entry. The instruction **VIEW KEYWORDS** will show all the keywords that are assigned to the open file.

It should be noted that keywords apply to entries, not to messages. Thus one cannot keyword a message and have that keyword apply to whatever files the message appears in. Also keywords do not move with an entry when that entry is **MOVE**d or **FILE**d into another file. This makes keywords somewhat awkward to use and may explain why they were used less than was expected.

2.14.8 Route

At CINCPAC the LDMX distributes message to the Directorate level. All messages for the J3 staff are delivered to J3's Administration office, J301. J301 must go through approximately 700 messages a day, assigning action, making distribution copies, and filing copies into appropriate subject files. This process is very time-consuming (approximately 4 hours in the early morning, plus additional time distributed over the working day) in the paper system. The more common messages that arrive every day (e.g., weather messages, FBIS) are easily recognized, but assignments must be made on them one at a time. Other messages require looking in an index file (primarily based on subject keywords) to ascertain the distribution.

When SIGMA was first delivered to CINCPAC, it was not much of an improvement over the manual method of distributing messages. Although classes of messages could be easily extracted from the file with **RESTRICT**s, the **ACTION**, **FORWARD**, and **FILE** operations each still had to be done on a message-by-message basis. Only **DELETE ENTRY** could be applied to the whole set of messages. J301 asked for a single operation that performed the **ACTION**, **FORWARD**, **FILE**, and **DELETE** commands and could be applied to a list of entries. The resultant **ROUTE** instruction also tried to capture the sense of the keyword index file, as explained below.

The **ROUTE** instruction takes as arguments an entry list and the name of an existing text-object which we call a *Route List*. As described in section 2.15, text-objects can hold any arbitrarily formatted text. To be a Route List, the text-object has to conform to a very rigid format as shown in Figure 2-9.

The names used for Route Lists can be the keywords from the manual index file or, as was done at CINCPAC, a separate text-object can be used to cross-index between keywords and Route Lists. Each Route List is a separate distribution pattern. J301's normal style of distribution with SIGMA is to **RESTRICT** based

Table 2-2: Selector attributes

From <STRING>	(e.g., From "J301" or From "FLEWEACEN")
Action <STRING>	(e.g., Action "J3")
Subject <STRING>	(e.g., Subject "Gale Warning")
Keyword <STRING>	(e.g., Keyword "Cambodia")
By <USER ID>	(e.g., By J301)
<SECURITY>	(e.g., Secret or SSSS)
<PRECEDENCE>	(e.g., Routine(R) , Priority(P), Immediate(O), Flash(Z))
<MESSAGE TYPE>	(AUTODIN, Memo, Note)
<MESSAGE PHASE>	(Preparation or Transmitted)
<RESPONSIBILITY>	(Orig, Action, or Cog)
DTG <DTG>	(e.g., DTG 021745Z OCT 77)
Before <DTG>	(Before and including this DTG, e.g, Before 021745Z OCT 77)
After <DTG>	(After and including this DTG, e.g., After 021745Z OCT 77)
Around <DTG>	(Within 24 hours of the DTG, e.g., Around 021745Z OCT 77)
<Entry Number>	(e.g., 17)
Before <Entry Number>	(Before and including this entry, e.g., Before 17)
After <Entry Number>	(After and including this entry, e.g., After 17)
Around <Entry Number>	(5 entries above and 5 below, e.g., Around 17)
Incoming:	a message that has been sent to the user.
Error:	an error condition has been detected. A brief description of the error is provided in the entry.
For_Action:	a message sent to this user for his action with ACTION.
For_Info:	a message sent to this user for his info with FORWARD.
For_Chop:	a message sent to this user for his chop with COORDINATE.
For_Releas:	a message sent to this user for his release with RELEASE.
Chopped:	a message returned as either Chopped(Y) or Chopped(N).
File_Copy:	an entry for a message that has been placed in the file.
Back_Copy:	a comeback copy of a released message.
Retrieved:	a message retrieved from Archive.
Deleted:	entries deleted from the open file during that work session. All deleted entries are destroyed after the file is FINISHED.
Recent:	newly arrived entries in a file since it was last opened.
Examined:	all entries in the file which the user has displayed.

```
          * Any text between asterisks is ignored. *
        * This is useful for commenting on the Route list *
ACTION:     <user name>.

FORWARD:    <list of user names separated by commas>.

FILE:       <list of files separated by commas>.

DELETE:     <YES or NO>.
```

Figure 2-9: Format of a Route List

on certain criteria, **ROUTE** all the entries selected with the appropriate Route List (which deletes the entries), !!*BACKUP ONE*!!, and repeat for the next class of messages. After all the easily classed messages are handled in this way, J301 reads the remaining messages one at a time and **ROUTE**s each using the most appropriate Route List from the index. The total process takes about one-fourth the time of the manual system.

2.14.9 Other Operations on Files

Most of the operations that apply to messages (e.g., **COPY**, **FILE**, **FORWARD**, **ACTION**, **PRINT**, **READDRESS**, **REPLY**, **DISPLAY**) apply as well as to file entries. The coordination functions (**CREATE**, !!*COORDINATE*!!, !!*CHOP YES*!!, !!*CHOP NO*!!, !!*RELEASE*!!) are the principal exceptions.

For large files it is often a very slow process to scroll through the file to move a particular entry into the Display window. If the entry number is known, the user may execute the instruction

```
††FIND <entry number>
```

and SIGMA will automatically make the specified entry CURRENT and put it on screen. **FIND BOTTOM** and **FIND TOP** move the contents of the screen to put the first entry or last entry on screen. These apply for the selected subset or the entire file, whichever is the active view of the file.

If the user does not know the entry number he seeks, but does know some text string in that entry that makes it distinguishable, he may execute a **FIND STRING**. SIGMA will scroll the file to the first occurrence of that string. To apply it again to find the next occurrence, the user must move the cursor down one entry, hit !!*HERE*!!, and then type in the instruction again. This difficult sequence limits the utility of this instruction.

2.15 TEXT-OBJECTS

Text-objects are arbitrary pieces of text that SIGMA allows to be created, stored, and used wherever text is appropriate. An addressee list might be a text-object. Route Lists (see section 2.14.8) are text-objects. When several users prepare individual parts of a message body (such as a SITREP), these parts can be generated as text-objects and then easily assembled into the message. In fact, text-objects can be used to prepare any written material, regardless of whether it is used in message processing. They are used for preparing briefings, reports, letters, or whatever.

When a text-object is created, its name and classification must be specified. There are two ways to create a text-object:

1. It can be done with the typed **CREATE TEXT** instruction. This causes an empty text-object to appear in the Display window, ready for the user to enter data.

2. The user can also generate a text-object from already displayed material by bracketing the text with !!*HERE*!!s and executing a typed **COPY TEXT** instruction with the name to be assigned to the object. In this case the new text-object is created and stored without **DISPLAY**ing it, and is assigned the same security level as its source.

If the user wishes to edit a text-object he simply **DISPLAY**s it, makes the changes he wishes, and !!*FINISH*!!es it.

Text-objects may be **PUT** into message fields, comments, or other text-objects. They, of course, can be **PRINT**ed, **DELETE**d, and **VIEW**ed. Standard editing operations such as **FIND STRING**, !!*UPDATE*!! (reformat), **PICKUP**, **MOVE**, and **COPY** apply as well. One can **GET** another user's text-object, which gives him his own copy. It will not reflect future changes made to the original text. The *Text-Object Directory* shows the names of all text-objects belonging to a user.

There are two instructions unique to text-objects. !!*SAVE*!! stores the currently open and displayed text-object into the disk file without closing the object. This operation is provided purely for protecting the user from system crashes. If the system (TENEX or SIGMA) halts for some condition, the user generally loses whatever editing he has done on the open objects. If he was preparing a ten page message, this could be very frustrating. To minimize the potential of losing work, the user may !!*FINISH*!! the message every so often, which will update the master copy of the message. But in this case the user will not be able to return to editing until he can display the message again, which cannot occur until the master copy update is complete. This can take several minutes on a heavily loaded system. If, on the other hand, he were to write the message body as a text-object, he could execute !!*SAVE*!! whenever he wanted to store his work and continue.

The other special operation unique to text-objects is reclassification. SIGMA does not allow the reclassification of messages, files, or selectors because these are complex structures in SIGMA's internal representation, and in the SIGMA security model the "trusted job" is not capable of handling them (see section 2.10, page 2-13). Text-objects are just text strings and are simple enough that the "trusted job" can process them. The trusted job is involved since it is the only process in the SIGMA security model which can pass data from a higher security level to a lower one.

When a user wants to **RECLASSIFY** a text-object he specifies the new security level. SIGMA presents each page (screenful) to the user and asks for a !!*YES*!! or !!*NO*!! confirmation. This is intended to ensure that the text being reclassified is exactly the text the user wants reclassified. This requirement comes from the basic premise of the SIGMA security model that the bulk of SIGMA code cannot be trusted (only the "trusted job," the operating system, the terminal, and the user can be believed).

2.16 ALERTS

The *Alert* facility calls a user's attention to important new messages as they arrive. He does this by defining a standard selector with the special reserved name *ALERT_SELECTOR*. This selector defines the criteria that an incoming entry must meet to be considered an Alert. As entries arrive for a logged-on user's Pending

file, they are automatically compared against the ALERT_ SELECTOR. If one meets the ALERT_ SELECTOR criteria, it is considered an Alert and it is appended to a special object called the *Alert List.*

The number of Alerts that have arrived since log on is shown on the Flash line next to the notification of new Pending file entries. Thus the flash line might say "Alert:1 Pend:7." Each time an Alert arrives, the bell is sounded.

To see his Alerts a user presses the function key !!*ALERT ON/OFF*!!. This puts his Alert List into his View window. An Alert List looks like a SIGMA file, except that there are two blank inverted video spaces where the entry number normally appears. Each Alert that has arrived shows as an entry in the Alert List. Thus the Alert List looks like the contents of the Pending file that has had the following applied to it:

```
††RESTRICT WITH ALERT_SELECTOR AND RECENT
```

The user may do a !!*HERE*!! on an Alert List entry and a !!*DISPLAY ENTRY*!!, which will cause the referenced message to be displayed. The user may also execute the **VIEW**, **COPY**, **REPLY**, **PRINT**, **ACTION**, **FORWARD**, or **FIND STRING** instructions for the indicated entry. The user may not execute any instructions that require the actual Pending file or the concept of a current entry such as !!*DELETE ENTRY*!!, !!*DISPLAY NEXT*!!, **ROUTE**, **COMMENT**, or **KEYWORD**. **AUGMENT**, **RESTRICT**, **BACKUP ALL**, and **FIND ENTRY** also do not have meaning for an Alert List.

If the Alert List is being Viewed when a new Alert arrives, the Alert List is updated dynamically and the user sees the new entry added at the bottom. SIGMA does not allow the Alert List to get larger than 10 entries. Any additional Alert arriving after that will push out the first entry in the Alert List. The user can clear his Alert List at any time by executing the **RESET ALERTS** instruction. This also resets the Alert count to 0. Since each Alert that arrives also shows as an entry in the user's Pending file, the Alerts that are removed are not really lost.

To remove the Alert List from the screen, the user may push !!*ALERT ON/OFF*!!, which returns the screen to its previous state (including the View window), or !!*CLEAR VIEW*!!, which will clear the View window and assign the space to the displayed object.

2.17 MISCELLANEOUS OPERATIONS

There are a handful of other operations available to the user which round out the SIGMA system. Most of these can be executed at almost any time during a session.

2.17.1 Log Off

To end a session on SIGMA the user executes the **LOG OFF** instruction. This instruction always requires confirmation. It !!*FINISH*!!es all open objects, making whatever changes are required to update the database. When **LOG OFF** is complete, the terminal is cleared, leaving the top line to say

```
TERMINAL FREE - previous user has logged off
```

2.17.2 Identify

In the Command Center and certain other locations, office codes stay logged on 24 hours a day, but individual users change. The **IDENTIFY** instruction lets a new user assume the role of the logged on office. This is done so that accountability for actions can be traced to the proper person.

2.17.3 Printing

SIGMA allows the user to print on paper any object or directory that he can access, by the typed **PRINT** instruction with the name of the object to be printed. In addition, function keys are provided to print whatever object is in the Display or View window. Each printed object is preceded by a header page which identifies whose printout it is, what its classification is, and a security disclaimer.

For MME seven Versatec printers are distributed around the user offices. Each terminal has an associated primary printer and a secondary printer. **PRINT** instructions executed at a terminal cause hardcopy to appear at the primary printer, unless that printer is not functioning (power off, out of paper, out of toner, etc.), in which case it prints at the secondary printer. If the secondary printer is also not operational, the Computer Center line printer is the ultimate fallback unit. Initially security procedures dictated that if a printer was down and SIGMA switched its output to the secondary printer, the change back to the primary unit required manual intervention (presumably by the System Security Officer). This procedure was found to be entirely too awkward to be practical, so the algorithm was changed to automatically try the primary printer first for each object to be printed. Unfortunately, SIGMA has no convenient way to inform the user when his printout is switched to the backup printer (see section 4.12.9 on page 4-87).

2.17.4 System News

As mentioned in section 2.4, page 2-3, a user is presented the System News in his View window when he first logs on. If a user wishes to see that news again later in his session he may execute the typed SYSTEM NEWS instruction.

2.17.5 System Status

This typed instruction presents in the user's View window a list of the other users who are currently logged on and at what security level they are operating.

2.17.6 View Display

The !!*VIEW DISPLAY*!! function key puts into the View window whatever object is currently in the Display window.

APPENDIX

This article originally appeared in *AFIPS Conference Proceedings* Volume 48 of the 1979 National Computer Conference. It is reprinted here with permission of the AFIPS Press, Arlington, Va.

SIGMA—An interactive message service for the Military Message Experiment

by ROBERT STOTZ, RONALD TUGENDER and DAVID WILCZYNSKI

USC/Information Sciences Institute
Marina del Rey, California

and

DONALD OESTREICHER

Xerox Corporation
El Segundo, California

MME OVERVIEW

The increasing sophistication of military systems and decreasing time frame for making decisions make it essential to provide the military commander better quality information faster. With today's technology, messages can traverse several thousand miles in fractions of a second, but hours are lost at either end, both in entering the message into the communications system and in delivering it to the person who can act on it. Even after the message is delivered, an officer acting on it requires background information to formulate a proper response. More often than not, that information is available only after time-consuming searching through ponderous files. The response is usually an outgoing message which must be coordinated with other people, many of whom are not in the immediate vicinity of the message drafter. Hand-carrying the draft to these people slows the response still further. In times of crisis this system can easily become overloaded, throwing the entire operation into disarray.

This message management problem seems an excellent candidate for automation. Users of the ARPAnet have had a form of on-line message service for more than seven years. There is no question that the technology exists, but whether it will be cost-effective in the military environment is not so clear.

In December 1975 the Defense Advanced Research Projects Agency (DARPA), Commander Naval Telecommunications Command (NAVTELCOM), Commander Naval Electronic Systems Command (NAVELEX), and Commander-in-Chief, Pacific (CINCPAC) signed a Memorandum of Agreement[6] stating their intention to conduct an experiment at CINCPAC Headquarters whose express goal was to "evaluate the utility of interactive message service capabilities in a military environment." The experiment is called the Military Message Experiment (MME).

To have the military conduct an experiment of this sort is highly unusual. More traditionally the user community would state their requirements in a Request for Operational Capability (ROC), which is interpreted and converted by some agency of the service into a system specification in the form of a Request For Proposal (RFP). The RFP is subject to further interpretation by the various contractors, first in their proposals in response to the RFP and later in the implementation by the winning contractor(s).

Although this procedure has apparently served well for procurement of more traditional systems, experience indicates it has been less successful in the field of computer automation, especially in data management systems like a message service, where the requirement is seldom well understood and the many levels of interpretation between the ROC and the final system lead to products poorly suited to the real requirement. The implications of a particular system design are subtle; if some aspect is inappropriate, it is often virtually impossible to change. ARPAnet experience has shown that the effectiveness of a message service is strongly dependent on ease of access to the system, how often people look at their messages, how "official" such messages are considered to be, *ad infinitum*. Further, the message service is often just the tip of the iceberg as users begin to understand the capabilities the system offers outside pure message processing. If the message service is used extensively, it intimately affects individuals' work style in ways that are difficult to predict. Thus it is very risky to try to specify "requirements" when replacing a paper, pencil and typewriter world with modern "office automation" tools. Attempts at management information systems have shown it is particularly difficult to provide a user interface that is acceptable to high-level managers, so it is not even clear precisely who will sit at the terminals or how these people will interact with senior officers.

The MME is a computer-world equivalent of a "fly-before-buy" test of a message service in an operational military environment. The test is relatively small and inexpensive compared to the cost of multiple installations of a "production" system. The hope is the experiment will provide

enough experience and understanding of interactive message handling that subsequent production systems will be successes rather than expensive lessons in how not to automate the process. In June 1977 the U.S. House Appropriations Committee put a moratorium on virtually all new development of "message systems" by the Department of Defense until results from the MME can be evaluated.

The MME is being conducted at the CINCPAC Headquarters, Camp Smith, Oahu. The test community is approximately 100 officers and staff personnel in CINCPAC's command center and "operations" directorate (called J3). Twenty-four video display terminals are provided for user interaction with the service. Seven printers are located throughout the headquarters for local hard copy. The host processor is a PDP-10 (KI processor) manufactured by Digital Equipment Corporation running the TENEX operating system developed by Bolt, Beranek and Newman.[4] The PDP 10 connects to CINCPAC's AUTODIN (AUTOmated Digital Information Network) terminal computer, called the LDMX (Local Digital Message eXchange). The LDMX supports the current message handling service at CINCPAC; it prints copies of all CINCPAC's incoming AUTODIN messages and accepts outgoing messages through an optical character reader, sending them to their specified addressees via AUTODIN.

TECHNICAL RESPONSE TO THE MME

The message service being used for the MME is called SIGMA, developed at the University of Southern California's Information Sciences Institute especially for this experiment. As such it is an "experimental" system to be used in an "operational" military environment to test the effectiveness of a "secure" "interactive message processing" service. Consider some of the issues implied by these terms.

"Experimental"

The primary purpose of the MME is to determine the effectiveness of interactive message processing in a military environment and to provide a technology-transfer path to apply the knowledge and techniques gathered to future generations of military systems. Many design philosophies are implied in such a context. The system developed must be flexible enough to adapt to a changing understanding of the problem and additional requirements imposed by the user community. It must concentrate on issues of functionality and suitable user interfaces. This does not imply that other issues such as sizing and performance are not important—the system must still be responsive and large enough to support a meaningful experiment—but the system need not be cost-justified itself, merely sufficient to gain understanding of the functional and cost issues. And, perhaps most important, the system must be highly instrumented to allow collection of various data reflecting the manner in which users operate the message service. Analyses of these data allow evaluation of user performance and usefulness of service facilities, and provide a further understanding of the potential role of interactive message services in the military environment. All of these factors were considered and respected in the development of SIGMA.

"Operational"

To gain the most accurate picture of the MME's potential impact on the military community, an early decision was made to perform the experiment in an actual military environment, the CINCPAC headquarters. Since CINCPAC already has an effective manual message system whose use is well understood by its personnel, SIGMA presents a message processing model which is intuitively similar to the existing manual one in order to gain early user acceptance. This decision implied choosing terminology which matched standard military usage,[5] and providing functions which, where possible, were similar to the manual ones. Since the military users operate the on-line system as only a part of their normal jobs, SIGMA has been designed to be highly self-instructive.

"Secure"

An important requirement for the SIGMA service is that it meet military security specifications. Although this test system will be operated only by personnel classified at Top Secret, it is a test objective that the message service address the multi-level security issues identified by previous research.[3,1] To satisfy this objective, SIGMA implements a security model which behaves as though SIGMA were running in such a "provably secure, kernel-based" operating environment; this model is described in Reference 2. Although the provably secure environment does not yet exist on TENEX, SIGMA's emulation of it allows the users to interact in a manner virtually identical to that they would encounter if SIGMA actually ran in the secure environment.

"Interactive message processing"

SIGMA has been designed to be a "complete" interactive message service for CINCPAC. Its user interface responds to the needs of computer-naive users in several ways. Since this community will not receive much special training in SIGMA's use, an online Tutor and Help facility has been designed to take on the bulk of this responsibility.[7] The Tutor, like the rest of SIGMA, takes advantage of the specially designed MME terminal,[8] which features multi-window displays and two-dimensional editing.

The command formats are defined by SIGMA's Command Language Processor (CLP). The user instructs SIGMA through a set of function keys or by typing commands in a predesignated "command window" on the screen of the MME terminal. The CLP parses and interprets these instructions; it is table-driven so instructions may be added or modified easily. The CLP expands commands and parame-

ters that are only partially entered and corrects misspelled words to the degree that it can, based on the user's personal directory of named objects as well as the command table. A Prompt facility is provided which allows the user to ask about required parameters for a given command without losing any of his operational context. This powerful command language interface is an essential ingredient in providing the user the highly supportive environment needed for users with little or no experience with computers.

SIGMA—APPLYING INTERACTIVE TECHNOLOGY TO MILITARY MESSAGE PROCESSING

In common with other styles of mutual communication, message processing has distinctly cyclic characteristics—a message is initiated; its contents are refined from draft to final form; it is approved and sent to its intended recipients; a recipient reads it, perhaps forwards it to colleagues; the information contained within invites (or requires) a reply; the reply is initiated, and the process repeats. Message processing as practiced at CINCPAC differs from other styles primarily in its highly formal nature, with guidelines governing nearly every aspect.

The entire message processing task can be roughly separated into three areas of closely related activities: Message management, incoming message processing and outgoing message processing. As implied by the cyclic nature of the communication process, none of the activities in one area is disjoint from those in other areas. For the purposes of presentation, however, the facilities of the SIGMA message service are described according to these three areas.

Message management

A message service must facilitate all phases of message management. The following sections summarize SIGMA's support in this area.

Messages

Messages are SIGMA's fundamental concern. They are composed of a diverse set of fields; a field's contents depends on its type. For example, a "TO" field contains a list of addressees, a "TEXT" field contains a succession of uninterpreted paragraphs, while a "SUBJECT" field can only have a single line of text (of arbitrary length). Although AUTODIN traffic is its primary focus, SIGMA also supports formal in-house communications (*Memos*) and informal messages (*Notes*). They differ slightly in the fields they contain and the ways in which SIGMA processes them.

Folders, entries, and selectors

Folders are the users' basic mechanism for organizing and storing collections of messages in SIGMA. They contain *entries* which are pointers to messages. A folder entry, an abstract of its message, contains information such as the message's precedence, security, sender, type and subject, which are considered by SIGMA as attributes of the entry. An entry for an incoming AUTODIN message might look like

```
22 R UU Auto 042222Z DEC 78 From:
        JCS WASHINGTON DC INCOMING   Act: J3
        Subject:  AIRCRAFT INFORMATION
```

The above entry is number 22, with routine (R) precedence, unclassified (UU) security, AUTODIN type, whose date/time is 042222Z DEC 78, etc. When a folder is DISPLAYed (the capitalized part of verbs are SIGMA commands), a numbered list of entries is put on the terminal's screen.

For use in commands, entries from folders can be identified in three ways: 1) By their number, 2) by default to the current entry, 3) by HEREing the entry number, i.e., putting the terminal cursor into an entry and depressing the "HERE" key. With this scheme most folder entry commands—DISPLAY, DELETE, FORWARD, etc.—have three forms. Using DISPLAY as an example, they are

```
DISPLAY ENTRY entry-number
DISPLAY ENTRY
DISPLAY NEXT ENTRY
```

The first is a typed command. The second is a function key which can take a HEREd entry; without one it will display the current entry. The last is also a function key. Some commands like DELETE and FILE (which copies entries from one folder to another) take entry lists, in addition to simple entry numbers, as parameters.

As objects themselves folders can be CREATEd, DISPLAYed, DELETEd, RESTOREd (the inverse of DELETE), and FILEd into. In addition, user directories of folders can be VIEWed and assuming access is permitted, folders can be shared among users.

Often a user wants to extract from a file a class of entries which have some uniform characteristics. For example, he might wish to work on his messages according to their precedence. SIGMA provides *selectors* for this task. Selectors are boolean expressions composed of attributes of entries. When applied to folders they act as filters, returning lists of entries whose members satisfy their criteria. When the user has a folder displayed he can use the two selector commands, RESTRICT and AUGMENT, to change his display to exactly those entries he has selected. Thus after DISPLAYing a folder, a user can see all secret entries from CINCPAC by typing the command

```
RESTRICT SELECTION SECRET AND FROM
CINCPAC
```

The user can AUGMENT his display in a similar manner. If he now wants to add to this display those entries whose

precedence is routine, he types

AUGMENT SELECTION ROUTINE

AUGMENT and RESTRICT commands are "stacked" so that the user can always back up to his previous display using the BACKUP function key.

At any point during a sequence of RESTRICTs and AUGMENTs the user can CREATE a named selector which reflects the logical "ANDing" (for RESTRICT) and "ORing" (for AUGMENT) which led to the current state. In addition, he can CREATE a named selector directly by typing in the boolean as an additional parameter. DELETE, RESTORE, and VIEW commands apply to named selectors. A user's directory of named selectors can be VIEWed, and if access is permitted selectors may be copied from other users.

The richness of entry attributes makes selectors easy to use for creating relevant folder displays. SIGMA commands that operate on entries apply only to those entries currently in view, i.e., selected. So if entries 5, 15, 22, 27 were selected, four uses of the DISPLAY NEXT function key would step through only those messages.

Comments

Message fields and folder entries may be annotated with arbitrary text strings by means of the COMMENT command. Comments are identified by the user making them and have additional access properties; they can be public, private (to the commentor), or restricted to a named user. Comments are created by pulling a HERE in the desired message field or folder entry. The COMMENT command is then entered with the access specification and in response SIGMA will create a new field ready for editing.

Editing and text objects

Fields can be edited in two ways—by modifying the display with local editing functions provided by the terminal, and by various SIGMA commands. Suffice it to say the terminal provides a full complement of editing capabilities.[8] In addition to those capabilities, SIGMA has its own editing commands. The PICKUP function key command deletes the characters between two HEREs, putting them into an unnamed buffer. The PUT function key inserts the contents of the same text buffer at the current cursor position. The MOVE function key, a composition of PICKUP and PUT, moves the text between the two HEREs to the current cursor position. COPY is the same as MOVE except the characters are not erased. These commands give the user the capabilities to erase, move and copy large amounts of text conveniently.

Text that is to be reused in messages or commands can be created and stored as a named *text object*. A text object is nothing more than a series of uninterpreted paragraphs, and can be CREATEd and edited using all the capabilities described above. A user directory of text objects can be VIEWed; they can be DISPLAYed, DELETEd, RESTOREd and, if the access is correct, copied from other users. Named text objects can also be used in conjunction with the PICKUP and PUT commands.

The content of a text object is unrestricted. No semantics are applied until it is put into an interpretable field. Once it is there, SIGMA acts on it depending on the type of field into which it was put. For example, text in an address list is checked for legal user names; when put into a field in which multi-paragraphs are allowed, the text is formatted.

SIGMA has not explored all the potential of text editing. So while it has a FINDSTRING command, it doesn't have one for text substitution. The experimental use of SIGMA at CINCPAC will provide feedback related to the adequacy of our editing model.

The SIGMA display

SIGMA divides the MME terminal screen into four windows. The FLASH window at the top of the screen contains three lines. The first is updated every minute and gives general operating information, time of day, and so forth. The FEEDBACK line tells the state of SIGMA processing and conveys error information. The STATUS line will be described below. SIGMA commands are typed into the two line COMMAND window below the FLASH window.

The remainder of the screen is the user's working space, which may be occupied by one or two windows. When a user DISPLAYs a folder, text object, or message, the object is "opened" and put into the EDIT window. If another object of the same type is already opened, then it is FINISHed, i.e., stored away with all new edits saved. If the open object is of a different type, then it is moved off the screen, though still opened, to make room for the newly DISPLAYed one. The STATUS line names all the open objects with the first name on the list identifying the one currently on the screen. Three function keys, SHOW FOLDER, SHOW MESSAGE and SHOW TEXT, can be used to put the appropriate object back into the EDIT window.

The VIEW window shows objects which the user names with the VIEW command. It is cleared by the CLEAR VIEW function key. This window is not editable and is used only for reference (text can, however, be copied from it). It is shown at lower intensity to distinguish it from the EDIT window. The EDIT window occupies the full screen when nothing is viewed; otherwise, both share the screen.

This section has given a functional view of SIGMA by describing its objects and some of its legal operations. The next two sections will give a more structured view of the tasks which compose message processing.

Incoming message processing

In the military, formal AUTODIN messages are sent from the commander of an organization to the commander of

other organizations, never between individuals within the organizations. This practice requires the receiving command to determine the appropriate recipients within the command for every incoming message. Naturally the correct assignment of recipients is a critical part of the incoming processing task.

CINCPAC employs a content-based scheme to determine the correct recipients. The first stage of this process is implemented by the LDMX message processor. By scanning the header and selected fields of the message contents, LDMX makes a preliminary assignment to the top management level (directorate). Although LDMX is capable of more detailed assignment, CINCPAC chooses to allow its directorates to perform their own routing internally. Within the MMF target population (J3, the Operations directorate), this next level of routing is performed manually by an administrative office (J301). Using both catalogued tables of routing assignment and his specialized training, J301 scans each incoming message and determines its disposition. Such disposition can be any or all of the following:

Action	Typically each message is assigned to an *action officer*. He is responsible for any actions or response to be made by the J3 directorate, and is said to *have the action* for the message.
Info	In addition to a responsible officer, the contents of the message may be of interest to other officers as well. Such officers are said to receive an *information* copy of the message.
Readboard	Certain messages may be of interest to large groups within the directorate, and occasionally to all of J3. Such messages are placed in binders called *readboards*, which are then circulated through the directorate.

While the J301 assignment is generally accurate, it is neither complete nor infallible. An officer receiving a copy of a message may determine that other officers not designated by J301 should also receive copies. Occasionally the action assignment for a message is not appropriate; after seeing a message, an action officer may decide that another officer is better qualified to handle it and "sells the action" to him. Thus, the propagation of copies and/or action of a message may continue for several stages beyond the J301 assignment. Based on data compiled at CINCPAC, an average of 40 copies of a message is required to reach all its recipients.

SIGMA supports incoming message processing with a variety of facilities. They can be roughly divided into the three areas of delivery, reception, and redistribution.

Delivery

SIGMA was designed to merge naturally into the existing message processing milieu at CINCPAC. Through the special LDMX interface, SIGMA's Reception Daemon receives the text of incoming AUTODIN messages, parses them, builds the SIGMA internal representation, and stores the resulting SIGMA-formatted messages in its message data base. To allow methodical retrieval of messages by arrival times, SIGMA also places an entry in a special folder called a *Date File*, a new instance of which is created each day to contain entries for all AUTODIN messages received during that day. A user can then see an index of all messages received on a particular day by simply DISPLAYing the corresponding Date File.

The high fan-out of incoming messages makes it impractical to provide a separate copy of each message to each eventual recipient. The scheme adopted in SIGMA was designed to minimize on-line storage requirements while still providing convenient access to messages. Messages are stored only once, in the central message data base. Each recipient receives not a copy of the message, but an abstract containing a useful subset of the message's contents. An instance of this abstract, called a *citation*, is created for each message transaction between users. Each citation sent to a folder causes a new folder entry to be appended (indeed, the terms *citation* and *folder entry* are often used interchangeably), a task performed by a special SIGMA background process, the Citation Daemon. A citation is small (approximately five percent the size of a message) and thus is much more economical to replicate than the full message.

All users access and modify the single copy of a message. Obviously such activity cannot occur unrestricted, or the integrity of the message contents and the users' intended changes could not be preserved. To allow such operations, a special scheme correctly assimilates parallel modifications, preserving both the consistency of the message and the users' intentions.[9]

Since messages flow into the J3 directorate constantly (approximately 1000 per day), the available secondary storage would soon fill unless appropriate steps were taken to reduce the number online. To make room for incoming messages, an archival scheme has been implemented. Using frequency of access as a rough guide, SIGMA moves inactive messages onto bulk storage (magnetic tape), from which users needing access to messages can request retrieval. Mechanisms are also provided to allow shorter retention periods for selected messages.

Reception

Once citations have been sent to a user, he must be allowed to see them, access their referenced messages, and dispose of them as he sees fit. These capabilities revolve around a repository for incoming citations, a special SIGMA folder known to each user as his *Pending File*. Analogous to a mail in-basket, a user's Pending File receives all citations destined for him.

Physically, a Pending File is implemented as a SIGMA folder, and can thus be manipulated by the wide variety of folder operations—DISPLAY of referenced messages, COMMENTing, cross-sectioning via RESTRICT and AUGMENT, etc.

Since all citations for a user are appended to his Pending File, he must eventually delete nearly all of them, lest he

exceed the folder size restriction (which is in excess of 6000 entries). This does not imply that a user must lose references to important messages, however, since a user may create an arbitrary number of other folders where he may FILE them.

Frequently, messages of great urgency to a user may arrive. In such cases the user would like to be notified immediately, rather than wait until he happens to notice it appear in his Pending File (which might be some time if he were DISPLAYing some other folder). To allow a user to specify criteria for incoming messages for which he wants immediate notification, SIGMA provides the Alert facility. The user activates the Alert facility by creating a SIGMA selector named ALERT_SELECTOR. If such a selector exists, each incoming citation is matched against it to determine if it meets the Alert criteria. If so, the citation is added to a special *Alert List*, the format of which is very similar to that of a folder, and the bell at the user's terminal is rung. The user can then display the Alert list without disturbing his open folder, and access any of the referenced messages in a manner similar to that for folder entries. In addition, a count recording the number of active alerted citations is maintained in the SIGMA flash line.

Redistribution

As explained in the description of the Delivery task, the routing provided by LDMX is not sufficient to reach all appropriate recipients. Additional routing is provided by the administrative J301 function, which supplies the bulk of the specific routing assignment, and by individual officers, who either supplement or correct the J301 assigments. SIGMA provides this flexibility by means of its redistribution facilities.

To effect the bulk routing assignment at the directorate level, SIGMA provides the ROUTE command. With this single command, J301 can specify action assignment, info distribution, and readboard creation for an entire group of messages. Using the RESTRICT and AUGMENT operations to select a class of similar messages, J301 can then perform a complete routing assignment for the whole class in a single step.

Individual officers needing to perform further redistribution have two more limited redistribution commands. The FORWARD command allows one user to send an information citation ("FOR_INFO") to another. The ACTION command is similar, but implies that the originating officer transfers the action assignment to the designee (by means of the "FOR_ACTION" citation). Additionally, the ACTION command causes an entry to be placed in the issuing user's *Action Log*, a special folder which contains a record of all action assignments he has made. Since CINCPAC wishes to keep a central accounting of action assignments, users normally share a single Action Log (via SIGMA's shared folder capability).

All redistribution commands account for the further distribution of messages by appending records to certain message fields. In each message a *Distribution* field records each user who has received an info citation, while an *Action* field records the full history of action assignments. Thus it is possible to ascertain all users involved in a message's redistribution by examining its appropriate fields.

Outgoing message processing

In the existing manual system, CINCPAC officers deal exclusively with so-called "record traffic." Even when the contents of a message are routine, the onus of representing an entire command's viewpoint adds a measure of importance. Consequently highly formalized procedures have developed at CINCPAC to ensure that messages transmitted from the CINCPAC organization have been thoroughly reviewed and approved by a responsible authority.

In addition to supporting an on-line implementation of the review/approval process, SIGMA has augmented the media of communication. In addition to the existing formal traffic (AUTODIN messages), SIGMA has added two new message formats, formal internal and informal.

- Formal internal messages (memos) are similar to content and form to AUTODIN messages, but the addressees are other SIGMA users. This provides CINCPAC personnel with a formal (recorded) medium to send official communications within the CINCPAC organization.
- Informal messages (notes) provide an off-the-record message medium for informal communication. Such messages, which are not reviewed or recorded, provide an alternative to face-to-face or telephone communication.

The outgoing message processing in SIGMA is roughly divided into four phases: drafting, coordination, release and transmission.

Drafting

During the drafting phase the original message is composed. The sources of the original contents of the body and various fields vary, depending on the type of message being prepared. SIGMA supports the following commands for message drafting:

CREATE	An empty message form is created, with blanks for the editable message fields. The contents of any desired fields must be filled in by the drafter.
COPY	This command, which requires an existing message as a parameter, copies all of the non-header fields into the new draft message. It is useful for pro forma messages which are sent frequently and whose contents are basically similar.
REPLY	This command also takes a message parameter, and creates a new draft in reply to the subject message. In this case, the addressees

are derived from the subject message, the subject is copied, and the referenced message cites are copied with a cite to the subject message appended as an additional reference.

Once he has created his draft message, the author has the following alternatives:

- He can save the draft for later use (via the FINISH function key). This can be done to hold an incomplete draft for later completion, or to save a pro forma message to make it available for later COPYing. To make later retrieval of such saved drafts convenient, SIGMA puts a citation referencing the draft in the user's Pending File, so its existence is remembered and it remains easy to access.
- He can send the message for review if it is a formal message. This process, called Coordination, will be described in more detail.
- If authorized, he can cause the message to be transmitted to its addressees. This will be described in the section on Release.

Coordination

Within CINCPAC there exists a formal procedure for review and revision of a message prior to its release. The drafter can request several other officers to review the message, make comments, suggest changes and give a general disposition regarding the message. This procedure, called *chopping*, permits CINCPAC to acquire a consolidated opinion from a cross-section of responsible officers before a message is sent.

SIGMA supports a style of coordination more general, although perhaps less flexible, than the manual CINCPAC procedure. The message drafter may designate any number of users in a field called the *Chop List*. With the special COORDINATE command, the drafter can specify that any or all of the users on the Chop List be requested to act as reviewers for the message (they are referred to as *coordinators*), causing a special "FOR_CHOP citation to be sent to each of the designated users.

A coordinator is notified of the drafter's request to review a message by receipt of the FOR_CHOP citation in his Pending File. He can display the message, and will see the drafter's most recent version. The coordinator can make comments or suggest revisions to the message; if so, the changes are not applied to the drafter's copy, but rather to a copy belonging solely to the coordinator. In deciding upon his changes he has access not only to his own version and the drafter's, but to other coordinators' as well.

When a coordinator decides that he in turn would like comments from other users (perhaps subordinates or other colleagues), he may further designate other coordinators. This "sub-coordination" is exactly analogous to that initiated by the drafter. In this case his sub-coordinators see his version of the message when they first display it.

When a coordinator has finished his review of a message he may indicate his global disposition of the message by means of the CHOP YES and CHOP NO function keys. These commands cause a "CHOPPED" citation indicating the appropriate disposition to be sent to the drafter (or higher level coordinator), who is thereby notified that this coordinator has finished his review.

During the coordination process, the drafter (or a higher-level coordinator) can monitor the coordination process by means of a status field, which indicates the progress of each coordinator. When coordinators have finished their reviews he can view their versions and note suggested changes and comments. He can incorporate changes by duplicating them in his own version or by copying the changed sections from the coordinators' versions. If he is not satisfied with the resulting message or wishes to elicit further review, he can initiate another coordination cycle, which will result in additional FOR_CHOP citations being sent. If the drafter is satisfied with the content of the message, he can initiate the Release process.

Release

Because of the formal nature of record communication, certain officers, designated *release authorities (releasers)*, are solely empowered to approve outgoing record traffic. SIGMA provides the same enforcement by checking each attempt to transmit a message against a list of authorized releasers (since the three different message formats have different levels of formality, a separate list is maintained for each).

When a drafter has determined that a draft message is ready for transmission, he must gain the approval of an appropriate releaser (unless he himself is one, in which case he can release it himself). He does this by using the COORDINATE command after designating the releaser's name in a special *Release* field, causing a "FOR_RELEASE" citation to be sent to the releaser. After receiving this citation a releaser has options similar to those of a coordinator. He can display the drafter's and coordinators' versions, seeing comments and suggested changes. In particular, he may examine the "chop" disposition of the various coordinators to determine whether he is satisfied that there is sufficient agreement among them. If he is not satisfied he can make his own comments and changes and specify CHOP NO, in which case a citation is sent back to the drafter. But if the message is in order and the releaser is satisfied, he can initiate transmission via the RELEASE command, whereupon the message leaves the preparation phase and is sent for transmission processing.

Transmission

When approved by a releaser, a draft message is prepared for transmission by the SIGMA process called the Message Daemon. First the draft is marked as transmitted; this prevents it from being further modified or transmitted again. A new message is then created to contain the transmitted ver-

sion of the draft message. Fields which are appropriate for transmission are copied from the draft; others which do not belong in transmitted messages (such as comments, chop lists) are omitted.

When the contents of the transmitted message are prepared, the appropriate transmission medium is determined. If the message is destined for AUTODIN, the message is sent through the LDMX interface to be transmitted to the AUTODIN network. If it is an internal message, the transmitted message (in internal SIGMA format) is entered into the SIGMA message data base, and "INCOMING" citations are sent to each of its addressees.

CONCLUSIONS AND USER REACTIONS

Our initial opinion after studying the CINCPAC environment was that an interactive message service could be extremely effective. The CINCPAC staff was enthusiastic about the possibilities and endorsed the experiment to the point that they were willing to serve as the test-bed for it. Now the experiment is underway and we are beginning to learn whether our optimism has been well founded.

Although at the writing of this paper formal results are not yet available, CINCPAC users have been using the service for six months. They have already asked for changes and extensions to the service; some, like ROUTE, have been implemented. As expected, the use of such a service is altering the style in which many officers operate.

Probably the most dramatic effect is on J301 who previously required seven hours to process the new messages that arrive overnight. Using SIGMA this process is reduced to less than an hour-and-a-half. Furthermore the feeling is the assignments made are generally better, primarily because the same assignment is made to entire classes of messages at once, thereby assuring uniformity.

Another group of users that has been heavily influenced by SIGMA normally get their messages from J301 about 9:00 AM, two hours after they come in in the morning. They have found that with SIGMA they can go directly to the Date Files for the day and, using Selectors, get the messages of interest without waiting for J301 to distribute them. They are also able to find messages requiring their action that have been assigned incorrectly to others, messages that they simply never saw before SIGMA was available to them.

There are still many improvements requested by CINCPAC users which SIGMA has not yet addressed. Indeed, the list is already large even at this early date:

- The ROUTE command was put into SIGMA in response to a direct request from J301. Other composite commands can be visualized. It would be nice to have a powerful facility for building such "macros" from existing commands. However, such a feature touches heavily on many difficult user interface issues.
- The ALERT mechanism is fundamental but acts only on incoming messages. Some users have expressed interest in a general facility based on a variety of different events.
- Users have expressed a desire for the ability to search the full message database with a mechanism like selectors. SIGMA has no model to support this expensive operation at this time.

Although the experiment is just beginning to collect useful information, it is clear that SIGMA is having an impact on the message processing at CINCPAC. SIGMA appears to be rich and flexible enough to support the goals of the experiment to gain insight for future military message systems. As the users become more involved with interactive message handling their awareness of its capabilities and potential is being sharpened and their requests for functional enhancements are more accurately based on realistic needs.

The injection of a research project, like SIGMA, directly into an operational military environment is an unusual event. This approach offers the military a more active role in developing relevant software for sophisticated applications. The MME effort is showing that the transition from the laboratory to an operational setting can be accomplished for such an experiment, which should dramatically shorten the normal technology-transfer path.

REFERENCES

1. Ames, S. R., and W. W. Plummer, "TENEX Security Enhancements," MTR-3217, MITRE Corporation, April, 1976.
2. Ames, S. R., and D. R. Oestreicher, "Design of a Message Processing System for a Multilevel Secure Environment," *Proceedings of the National Computer Conference*, AFIPS, 1978.
3. Bell, D. E., and E. L. Burke, "Secure Computer Systems Mathematical Foundations and Model," M74-224, MITRE Corporation, October, 1974.
4. Bobrow, D. G., J. D. Burchfiel, D. L. Murphy, and R. S. Tomlinson, "TENEX, a Paged Time Sharing System for the PDP-10," *Comm. ACM*, Vol. 15, No. 3, March 1972, 135-143.
5. Heafner, J. F., and L. H. Miller, "Design Considerations for a Computerized Message Service Based on Tri-Service Operations Personnel at CINCPAC Headquarters," Camp Smith, Oahu, ISI/WP-3, USC/Information Sciences Institute, September, 1976.
6. Memorandum of Agreement between Director, Defense Advanced Research Projects Agency (DARPA), Commander Naval Telecommunications Command (NAVTELCOM), Commander Naval Electronics Systems Command (NAVELEX), and Commander-in-Pacific (CINCPAC). Unpublished memorandum.
7. Rothenberg, J., "On-Line Tutorials and Documentation for the SIGMA Message Service," *Proceedings of the National Computer Conference*, AFIPS, May, 1979.
8. Stotz, R., P. Raveling and J. Rothenberg, "The Terminal for the Military Message Experiment," *Proceedings for the National Computer Conference*, AFIPS, May, 1979.
9. Tugender, R., "Maintaining Order and Consistency in Multi-Access Data," *Proceedings of the National Computer Conference*, AFIPS, May, 1979.

This research was performed for the Advanced Research Projects Agency under Contract No. DAHC 15 72 C 0308, ARPA Order No. 2223. The views and conclusions expressed in this paper are not necessarily those of any person or organization except the author(s).

PART THREE:

EVALUATION

3.1 LESSONS LEARNED

The Military Message Experiment was unique in many ways, for both the military and the research community. It was perhaps the first deliberate attempt to install a large experimental computer application at an operating military command expressly for the purpose of learning how (and whether) to build future production systems. The experiment did not attempt to justify the cost of automated message handling--that requires projecting design and operational costs for a system, estimating the operational benefits such a system provides, determining how many such systems will be produced, etc. It is only after the experiment is completed that a reasonable specification for that system can even be produced. An operational requirements document (ROC) for automated message handling is being written at this time by CINCPAC (Commander-in-Chief, Pacific) and EUCOM (European Command).

We learned a great deal during the experiment about what should be the proper functions of this kind of message system, but these lessons were only part of our education. The experimental results were affected more by several higher level issues than by the details of the message service operation. This part of the *SIGMA Final Report* is divided into the following major sections:

- High-level issues;

- Functional and design considerations for a message service;

- Lessons on development and operational environment for the experiment.

The previous parts of this report, which concerned the history of MME and the functions and design of SIGMA, are factual; this part, on the other hand, primarily contains opinions of the authors (all members of the ISI team that developed SIGMA), formed from our review of data abstracted from the user interviews, our own discussions with users, and other peripheral observations.

3.2 HIGH-LEVEL ISSUES

3.2.1 The Definition of Utility

According to the Memorandum of Agreement of its sponsoring agencies, the primary goal of MME was "to determine the utility of an interactive message service in a major military headquarters." But there is no uniform, objective yardstick for measuring "utility." Several consultants tried to define such a measure for the experiment but did not succeed, primarily because most of the parameters could not be quantified (e.g., quality of the end product, throughput speed, user satisfaction), and many variables in the operational setting could not be precisely controlled. Therefore, one of the best measures of SIGMA's utility is revealed in the subjective evaluation of the system obtained by interviewing its users.

As the experiment drew to a close, people from MITRE, CTEC, and the Navy extensively interviewed 50 users to elicit their opinions of automated message handling in general and SIGMA in particular. In addition, in March 1979, shortly after Exercise Power Play, 15 users were interviewed by Col. Clay Smith of the CINCPAC staff. Abstracts of the user interviews contain many illuminating comments; the users express a wide variety of opinions about the impact and utility of MME. With a few exceptions, the objections were directed toward specific deficiencies that could be remedied in a new system. The general climate of opinion was positive, but there were severe criticisms of system reliability and some individual reservations about specific functional deficiencies. The final reports from CINCPAC, MITRE, and the Navy should contain detailed analyses of these interviews.

3.2.2 The Value of SIGMA

We believe that SIGMA amply demonstrated the utility of automated message handling, both in everyday use and in simulated crisis, a judgment based on the fact that CINCPAC ultimately wanted to keep SIGMA, on the generally positive nature of the user interviews, and on the amount and type of usage SIGMA received throughout the experiment.

Approximately six weeks before the experiment was to end, CINCPAC sent a message (CINCPAC 1617172 AUG79) to JCS stating, "This headquarters considers it most desirable to retain the MME system past the experiment's conclusion." This statement was a reversal of CINCPAC's earlier position (as stated in CINCPAC 260136Z MAY79) that the system should be removed at the conclusion of the experiment on October 1, 1979. This reversal was ascribed to "significantly increased hardware reliability over the past eight weeks, improved system software, and increased user appreciation of the system's capabilities." We assume a future production message service would receive a similar endorsement if it provided the necessary reliability, presented a good user interface, contained equivalent functional performance, and was introduced with proper user-training procedures.

Our strongest indication of the usefulness of automated message handling was the sustained use of SIGMA not only throughout the experiment, but especially during the last few months when it was known the system would be disconnected and there was little external motivation for its use; more user hours (4,178) were recorded on SIGMA during August 1979 than during any other month of the experiment. If the system were not truly useful, usage would have dropped dramatically as users became aware the service would be terminated.

Our optimism about the usefulness of automated message handling is also encouraged by the reasonableness of extensions that users have proposed to enhance the system. Sections 3.3.2.26 and 3.3.2.27 discuss the proposed enhancements and the design implications of each. It is clear that as the users learned what SIGMA could do for them, they began to extend their own thinking about how SIGMA could be made more effective. This tells us that SIGMA did indeed provide a useful tool for doing their work better and faster, and that it became an integral part of their thinking about their jobs. Our conclusions about SIGMA's usefulness must be tempered by considering for whom and for what it was successful, and what parameters affect this success. The bulk of this section is directed toward this consideration.

3.2.3 Some Unqualified Successes

3.2.3.1 User interface

A good, consistent user interface is critical for the kinds of users we had at CINCPAC. With little computer experience, and only a rudimentary grasp of the details of message handling (a small part of his activities), the automated system user will not accept a system with baroque procedures that require much training. It is imperative that the system be natural, intuitive, and easy to use. This point was recognized from the beginning of SIGMA's development. For example, the terminal was especially designed to support highly responsive, two-dimensional "what-you-see-is-what-you-get" editing. A special study, performed at CINCPAC in July 1976 nearly a year before the system was installed, served to gather information as design input. Recommendations from this study were directed toward a variety of user interface concerns, including user vocabulary (for commands, message forms, and general system objects), instruction forms, screen formats, and user attitudes. The report *Design Considerations for a Computerized Message Service Based on Tri-Service Operations Personnel at CINCPAC Headquarters, Camp Smith, Oahu* [17] describes this study in detail.

It is hard to provide any quantitative results regarding the user interface of SIGMA. Although there was discussion of running "structured" tests (controlled experiments to evaluate individual user interface features) outside the CINCPAC community, these tests were never conducted. Our conclusion that the basic user interface was good is based on interviews and our own observations. There were no negative comments about SIGMA's user interface during the interviews. New users were able to do useful work with very little training. During the exercises, users from directorates other than J3 were introduced to the system, and though they did not understand the intricacies of the system, with only a few minutes training they could log on, display their files, display messages, and scroll through their contents. With a little more training the users were able to search the message database, create and edit text, and send messages.

We attribute our success to many factors: the "naturalness" of the terminal-based editor, the editor's responsiveness to the user, the sophistication of the Command Language Processor, the attention paid to presenting data in familiar forms, a system vocabulary that was familiar to the user community, and a philosophy of always presenting the user with as much contextual information as we could. These factors are critical to the success of future systems; while SIGMA did well, much remains to be learned.

3.2.3.2 Better access to information

SIGMA demonstrated to CINCPAC the benefits of improved communication through easier and more universal access to information, more accurate use of data, faster delivery of messages, and relief from some of the tedious work associated with text preparation and message handling. SIGMA selectors simplified access to information by allowing users to build and store their own criteria for locating messages in a flexible file system. Several offices and watch teams (e.g., JRC, Nuclear Operations) used prestored SIGMA selectors every day to scan incoming traffic for messages of interest to them. This allowed them to catch messages that the LDMX did not route to them--messages they otherwise would have missed. (Those offices that used SIGMA early in their shift found they saw their messages several hours before the paper system delivered them.) The Head of the Nuclear Operations group reported that by using selectors to cull incoming messages he was able to obtain an overview of what was going on within CINCPAC, a perspective he would never get from the paper system. The ease of sharing data in SIGMA also allowed users to look at the readboard prepared for J3, a feature that was extremely popular with the division chiefs.

Another form of information sharing was the automatic update of the Action and Information fields of a message. Whenever a new assignment was made to a message, these fields were altered to reflect it. Since all users shared access to the same copy of each message, this information was always up-to-date.

The ability to share access to comments on a single up-to-date file or message made it possible to use SIGMA to monitor and control user actions. For example, during the special CINCPAC-only rerun of Exercise Power Play, the Crisis Action Team actively managed their activity via SIGMA comments. A citation appeared in the Action Log for each message on which action was assigned. When the action officer had pertinent information to disseminate, he would append a public comment on the entry in the Action Log. The Executive Officer (XO) of the team monitored the Action Log; when comments indicated an action had been completed, he would move that entry to a "Completed Action Log." Through the Action Log an up-to-date status of the team's activities was instantly available to all users.

Shared access to text material also proved valuable. The text editing and word processing features of SIGMA were extensively used in preparing nonmessage as well as message material. One example was the preparation of the daily summary during the rerun of Exercise Power Play. Each member of the Crisis Action Team prepared his section of the report as a text object. When all sections were ready, the XO merely moved copies into the body of a prestored message, edited it for uniformity and clarity, and released the message.

The information contained in the individual sections of the daily summary was generally accurate (transcription errors were virtually eliminated), since much of it was copied directly out of the original messages.

In the manual message system, messages are kept in paper form for 30 days by J301. The Communications Center keeps messages on-line for 15 days and on tape for 90 days; after that they are microfilmed. It is a great ordeal to track down an old message in this system. Once SIGMA began archiving messages there was a simple and dependable procedure for retrieving all old messages. It was acclaimed as one of the best features of the system.

3.2.4 Limitations of Automated Message Handling Systems (AMHS)

3.2.4.1 Difficulties of user adaptation

It is clear that there are also drawbacks and inherent limitations in systems like SIGMA. In large part, these drawbacks have to do with the fact of life that users have to adapt to current computer systems, rather than vice versa. Some users will adapt more readily than others, while the dissatisfaction of those who do not will be reflected in their evaluation of the system. The best example is perhaps the display medium, the cathode ray tube (CRT). CRTs have some distinct advantages over paper, but they have disadvantages as well: they are often difficult to read, they are not portable, you can't write on them with a pencil, they have a limited working surface, and you can't leaf through their pages.

An automated message system is very different from a manual one. Models of manual procedures often do not translate well into automated environments--what is easy to do manually may be very difficult to automate (and vice versa). For instance, coordination of a draft message is most naturally a serial process on paper. Parallel coordination in SIGMA is certainly faster, but it requires new procedures and a different model of a draft message. Users without typing skills are seldom comfortable at a keyboard. The whole interactive style of giving commands with parameters in an artificial language is intimidating to many people. The longer an individual has been working in a particular style, the less willing he is to adopt a new way of doing business, which perhaps explains why senior officers were especially reluctant to try SIGMA. In any case there was ample evidence that it takes some people a long time to learn to use an automated message service.

3.2.4.2 Rigidity of automated systems

Besides the adaptation that users had to make, there are a number of deficiencies inherent in automated message handling systems. As mentioned above, paper can be a more flexible medium when scanning through a moderate amount of data. In addition, an automated system cannot substitute for face-to-face meetings when issues need to be discussed. It simply takes too long to conduct a written dialogue; it takes considerable skill to show emphasis or emotion in the written word; and there is a certain "permanence" about writing that can be inhibiting. The most severe objection to using SIGMA for coordination was the lack of personal encounters. Interestingly, this view was generally held by the junior officers who wanted to meet with their seniors to "explain the message," but not by the senior officers who perhaps saw automation as a chance to eliminate much of the personal interaction that occurs in staffing a message.

In addition, although automated systems can be very helpful in alerting an on-line user of the arrival of new messages, if the user is not on-line, the system has no way to "find" him and deliver the message. Furthermore, it is obvious that human beings can be more flexible than machines when flexibility is called for

(when it suddenly becomes impossible to do things in the usual way). Procedures that can be easily bypassed or altered in the manual system are often buried in the code of an automated service and are almost impossible to circumvent or change.

3.2.4.3 Particular limitations of SIGMA during MME

Certain limitations were not inherent in automated systems, but only a reflection of the circumstances of the MME. Everyone agreed that the lack of on-line access to all the reference material needed to process a given message was an annoying drawback. Although primary references are to other messages, other kinds of data are frequently relevant, too. Manuals, letters, operational plans, data of all sorts are pertinent; a system that can deal only with messages is deficient. Since some information may never appear in an on-line form (no matter what the system), this particular drawback may be an inherent limitation of automated systems. SIGMA did allow the user to manipulate arbitrary text objects and associate comments with messages and files, but several users remarked that it was awkward to have their message files separate from other documents.

Even when references were to on-line messages, it was hard to move quickly back to the original message. Mechanisms for directly accessing referenced messages and for the simultaneous presentation of multiple messages are high on the list of enhancements that should appear in any operational AMHS. SIGMA allowed internal memoranda to be produced and distributed on-line. However, since only a few of the CINCPAC staff had access to SIGMA, most memos were only in the paper files. Having some memos arrive via SIGMA was more of a hindrance than a help--no one wants to keep two sets of files. What few SIGMA memos were generated were generally printed and filed in the paper files. The lesson is clear. A user wants to receive, store, and retrieve all of his message traffic in a uniform fashion. The degree to which a message service must compromise this guideline will degrade its acceptability.

Response time is also a user interface consideration, one in which SIGMA was deficient. During the first year and a half, SIGMA's response time with moderate load was too slow for a meaningful experiment. With the installation of the KL processor in October of 1978, system response was satisfactory enough to proceed with full experimental usage. Even then, under heavy load, response time was marginally acceptable. This judgment of "acceptable" performance is somewhat arbitrary. No controlled tests were conducted to attempt to establish what is "acceptable." We relied primarily on our own experience and users' feedback.

If an operation can be done significantly better on-line than it can be done manually (e.g., **RESTRICT**, which extracts a subset of messages from a large file), the user is generally quite happy regardless of the time it takes. However, if it takes longer than he is used to (e.g., **DISPLAY MESSAGE**, which takes 5-10 seconds), he is not. It is not the absolute time required; it is more a matter of the user's expectations. SIGMA treats all operations the same, they are parsed by the same CLP, they are executed by the same Functional Module, and they draw their data from the same file system. Because SIGMA was built on a general-purpose time sharing system, there was little opportunity to tailor the operating system performance to the application. This was a restriction that we not only accepted but were comfortable with due to the experimental nature of the system. However, a production message service should be designed to meet predetermined performance goals, especially for the important, commonly used operations (e.g., display file, display message, next page). The operating system must be selected (or modified) to support these response time targets.

An aspect of the user interface that we considered but did not have time to explore was personalizing the service to individuals. Early plans (see [20]) called for a heavy emphasis on keeping a user model, monitoring the user's dynamic activity, and altering the way the system appeared to him as he became more fluent with it. This goal was viewed as being too experimental for the CINCPAC user, so plans were reduced to keeping a

static user model which could be manually modified. Even these ideas were later simplified. As a result MME users were given very little control over how the service appeared to them, an objection we generalized from what we heard in the user interviews.

3.2.5 Lessons Concerning the Service

MME taught us some lessons that are critical to successful installation of an automated message system. Like the user-oriented issues discussed in the previous section, these issues largely affect the usefulness of the service.

3.2.5.1 Reliability and availability

The most painful lesson we learned during MME is that reliability and availability of an AMHS are paramount. We had naively assumed that, since MME was an experiment, the SIGMA users could accept less that 100 percent uptime. Emphasis on solid operations and maintainable software occurred too late in the experiment, and therefore much of the system's potential was never truly tested. Many users were afraid to trust their time and vital files to SIGMA, using the automated system only for activities they could afford to lose. To be useful, an AMHS system must be ultrareliable. A crisis will not come to a stop while the computer is down.

Even with this attention to reliability, one can envision situations in which the normal service becomes unavailable to some or all of a user community. During wartime, it is probably infeasible to maintain the service level that can be achieved in peacetime operation; even if the service could be maintained, what if that facility is destroyed? MME did not investigate this issue, but the architects of future AMHS must address this problem. Pertinent research on the subject is being conducted by ARPA.

3.2.5.2 Integration of AMHS and the message exchange

SIGMA communicated with AUTODIN through the message exchange system called LDMX, a completely separate computing facility designed without any anticipation of communicating with an on-line message service. Creating an interface between SIGMA and LDMX proved to be a difficult task. The resulting interface was neither efficient nor easy to operate, and several functions were impossible (for example, LDMX could not deliver Top Secret traffic to SIGMA). LDMX facilities that should have been made available to SIGMA users (such as the Plain Language Address Tables) were not. Redundant processing took place (e.g., outgoing message formats were checked in both machines). Information available in LDMX was not passed to SIGMA (for example, linking of multipart messages). Feedback that a message was accepted for release to AUTODIN was delayed.

Some features will be very difficult to implement without much closer coupling between the message exchange and the AMHS. One prominent example is providing an on-line facility to update message distribution tables. These tables exist in LDMX, but there is no provision for remote access or editing of their contents. Ideally, the AMHS functions should be integrated with the message exchange functions.

3.2.5.3 Worldwide telecommunications procedures

After AUTODIN became operational, a set of standard operating procedures and formats were established to facilitate the input, routing, editing, and other handling required. These protocols were concerned strictly with that information needed to get the message to the proper AUTODIN terminal: type of message, precedence, addressees, classification, codes accepted, and message termination. This information was all

made a part of the message header, and since this information is computer-processed, the protocols are rigidly enforced.

However, a great deal more information is needed to properly handle messages at the recipient site. This includes such data as originator-proposed distribution (who should receive a copy of the message), the subject, references, passing and special handling instructions, a unique identifier, paragraph indications, individual paragraph classification markings, indication of multipart messages, identification of exercise traffic, downgrading instructions, and standard subject index codes. Unfortunately, the procedures and formats for these data are not clearly defined or rigidly enforced. Each service publishes its own guidelines (these are similar but not identical), while some organizations such as the State Department seem to have unique procedures. For manual handling of messages, this lack of uniform procedures is a nuisance; for automated message handling it is a disaster. SIGMA was severely limited in the service it could provide because of the difficulty in extracting this information from incoming messages.

This problem was most obvious in our attempts to locate the subject of an incoming message to be displayed in the file entries. The subject was supposed to indicate the content of the message and suggest to the user whether he should read it and how he should route it. Whenever the subject was wrong, the user was upset. The problem, however, was not with SIGMA, but with the lack of a uniform method for extracting the subject from the original message (if a subject was even given at all). Lack of formatting standards also limited SIGMA's ability to extract proper references, recognize exercise traffic, link multipart messages, or follow special handling instructions. Better standards alone are not enough; they must be enforced as well. Until all AUTODIN traffic is input through devices that will check the validity of this data, there is bound to be a large amount of human error on input.

3.2.6 General Questions Connected With Establishing an Automated System

As a message service is intended to provide communications among users, the benefits gained are directly related to how many people are on the system, who they are, and where they are located. An analogy with telephones is useful. The more phones (or terminals) on the system, the more useful the system becomes to everyone. In a message system, it takes a critical mass of users to contribute enough information to the shared database to make that database valuable. Some minimum number of users is needed in order to justify an automated service.

This phenomenon is amplified when the automated message service is not well integrated with the manual message facilities, as was the case with SIGMA. If an action officer wished to forward a message to two other officers, one on-line and the other not, he had to perform two completely separate operations. In most cases it was easier to handle both transactions manually. A better integrated system would reduce this problem.

Another factor closely related to the number of users is the number of terminals. The style of use is strongly influenced by whether or not a user shares his terminal. Obviously, if a user has his own terminal, he can access the system whenever he wants. However, terminal costs, system performance, the availability of ports, etc., may temper the ideal of one terminal per user.

The comments of some of the users interviewed made it clear that the number of terminals at MME (24) was below the critical threshold for CINCPAC. The precise minimum number is a matter of conjecture, though it was estimated by some at CINCPAC that 65 terminals would suffice to cover the J3 Directorate (containing approximately 200 people). It is not clear that covering a single Directorate would be the most effective use of that many terminals. Our guess is that 200 terminals would support automated message

handling for the entire CINCPAC Headquarters. As an AMHS becomes an integrated information/communication system, the distribution of terminals will no doubt go up.

If only a restricted number of terminals is available, as will surely be the case for any near-term AMHS delivered to CINCPAC, someone must determine their optimum allocation. As noted earlier, a user is directly affected by sharing a terminal. He will store essential data in paper files. If his job entails a great deal of this essential data, he will simply opt for doing his job entirely off-line; he will benefit little from the AMHS.

When only part of the user community has access to the on-line system, special procedures must be designed to support the integration of the manual and automated systems. For example,we might have had on-line users act as though everyone at CINCPAC were an on-line user. Then, whenever a message is forwarded to an off-line user, the system prints a copy of it with the recipient's name in the Communication Center and its delivery becomes the center's responsibility. Since no such procedure was tried during MME, we cannot say whether it would have been effective.

3.2.7 Policies and Procedures

So far in this part we have concentrated on the user's subjective reactions to the system. Now we focus on the fact that in the military each user employs the system within a fairly well defined set of policies and procedures.

Closely related to the issue of the number of terminals are the questions of specifying the users of the system, determining how often they can access the system, defining what they are allowed or required to do with the system, etc. The policies and procedures surrounding an AMHS are as important to optimizing the system's utility as the functional details. Especially important are the procedures relating to the interface between the on-line world and the hard copy world.

It is only through trial and error that optimum procedures can be discovered. This takes time, an organization willing to experiment, a system with flexibility, and a manager who has the authority and commitment to work out these procedures. Unfortunately, very little attention was paid to issues of how to best use SIGMA during the MME, so our results in this area are limited. Whatever automated assistance is provided, it is essential to devote appropriate attention to developing policies and procedures within the user community for the system's use.

Simple policies, such as how often a user is expected to check his messages or who is allowed to file messages into what files, determine the utility of an AMHS. For example, the policy of producing J3's daily readboard on-line uncovered one of SIGMA's most appreciated features since it gave division chiefs access to information they never had before. Although this readboard had to be prepared twice by the Command Center Watch Team, one hard copy and another on-line, if the MME printer had been of better quality the readboard could actually have been prepared entirely on-line and then printed for J3.

Changes in procedures often entail changes in various system parameters. It is therefore critical to have automated aids to change access-control parameters for the various objects in the system, to introduce new users, to change user parameters, to alter routing tables, to adjust security settings for users or terminal lines, etc. Since policies and procedures are prone to change and evolve, it is imperative that the AMHS not lock user procedures into its own code (even if this is done to "help the user"). For example, in SIGMA the automatic generation of a citation to the file ACTION-LOG on execution of the **ACTION** instruction is

buried in SIGMA code that performs this function as an adjunct to forwarding a message. The name of the instruction that calls this function, the citation type produced, and the name of the file to receive the citation are parameters of the SIGMA Command Table, which can easily be changed without writing new code. But if it were decided that action citations should be sent to two files, or that the **COORDINATE** instruction should send a citation to a Coordination-Log file, the system could not accommodate this without generation of new code and a full system release.

An AMHS is conceptually different from a paper system. Some of the most powerful features of an on-line service (e.g., ease of data sharing) are the most difficult concepts to understand. If an organization's procedures are to be tied to these complicated concepts, it must be recognized that it will take additional time to train users.

3.2.8 The Larger Goals of an Automated Message Handling System

An automated message handling system is much more than a delivery mechanism for getting a message from point A to point B. In fact, we believe message handling will be the base for a broad, general-purpose information/communication system of the future. To be useful, an AMHS must be a data management system and a word processing system as well. Furthermore, in order for it to realize its full potential, it must be a gateway into a host of other application programs, particularly retrieval of command and control data.

As such, an AMHS is an "administration" tool or, in more popular terms, an "office automation" system. Only when viewing the system in this larger perspective can one appreciate the significance of the many functions that should be provided. The AMHS must support a total organization whose different members will derive benefits from its various parts in very different ways. MITRE's results (see [13]) and users' remarks corroborate this finding. (Perhaps the best example is the **ROUTE** instruction, which was seldom used by anyone other than J301. But his role was vital to the message delivery function, and to him the instruction was essential.)

Since SIGMA was used as more than a message delivery system, it is not surprising that many of its underlying functions are not unique to messages; in fact, only eight of the fifty instructions that SIGMA executes are specific only to messages. Functions such as editing, printing, file manipulation, searching, annotation, help, tutoring, alerting, and status display appear in nearly all good on-line computing services. One of the compelling reasons for an integrated information/communication system (rather than many separate systems) is the way these functions can serve a wide range of data objects. One of the lessons we learned best is the necessity of looking at any future system from the widest possible perspective.

One of SIGMA's notable strong points has always been its attention to the real needs and preferences of its users. We argue that this attitude is indispensable at all levels of system philosophy and design. In fact, the necessity of finding out what these needs were is precisely why an experimental system was installed. One of our most important conclusions is that the next AMHS must be built in an evolutionary manner. Early models must be kept extremely flexible at the expense of performance or simplicity of programming, so they can be adapted to accomplish their goal; experience will tell how to introduce more rigidity in subsequent models. Though a system based on this mode of operation takes longer to build than a tightly coded system, the long-range effect is to deliver a better quality product--acceptable to a wider community of users. Although we feel that an AMHS has tremendous utility for military C3 environments, early delivery of an inflexible, insufficient system will do more harm than good.

A final point about the workings of an AMHS: we do not yet know how to integrate such a general-purpose service into a large organization. As an organization learns to use the functions of a given system, the

user's perception of his needs changes; many enhancements are suggested, new data objects are invented, and old ways of using a service are revised. Changes are introduced because system designers learn how to do the task better, see new benefits by extending the system's capabilities, and realize how certain local variations must be accommodated. The "evolutionary" approach we see as crucial is not easy to achieve, especially in a procurement environment designed to purchase tanks, airplanes, and guns. The Military Message Experiment was a significant step toward "evolving" an AMHS. We hope that the next step is taken from this solid footing.

3.3 LESSONS ON FUNCTION AND DESIGN

This section addresses those lessons that are specific to the functionality or design of an AMHS. Since the experiment was confined to a single automated message system, SIGMA, our examples are necessarily specific to that system. However, we have attempted to interpret the results as generally as we felt we could.

This section starts with a discussion of our opinion of the basic architecture of SIGMA. It follows with a wide variety of lessons about details of the message system, directed toward people who specify or design message systems.

3.3.1 Architecture

Section 4.2 of the *Sigma Final Report*, Part 3, discusses the architecture of SIGMA. The primary features of the architecture are shared access to a single copy of messages, files as a collection of citations to messages, shared access to a single copy of files, central data management, split between foreground-background processes, archival, and the intelligent terminal.

3.3.1.1 Shared access to a single copy of messages

The fact that the system keeps a single copy of a message and users share access to it provides many benefits, the most obvious of which is the cost saving in disk storage. Had we adopted the more conventional approach of providing each user his own disk copy, we would have required considerably more disk space. However, we feel the more important benefit is the inherent sharing of identical information. When a user accesses a message, he always gets the most up-to-date copy of "the" message. Pertinent information about that message (e.g., who has gotten copies, who has the action, appropriate annotations) is right there with the text; there is never an issue of having to send extra copies around to disseminate information about the message or of getting the "wrong copy" (e.g., the one without the annotation). This feature is especially important to the coordination process, where more than one person's version is involved and the message is constantly changing.

3.3.1.2 Files as a collection of citations to messages

Files (or folders, as we call them in Part 4) are collections of messages. But since messages are kept centrally, files necessarily contain pointers. In order to give the user enough information to recognize, select, and manipulate the messages, file entries contain only pertinent information. The power of extracting essential information into file entries is that it allows the user to perform the bulk of his manipulation of messages with this compact information; this is faster both for the user and for the system. The concept of files as a collection of pointers also makes the archive service work smoothly, since it permits the messages to be moved to tertiary storage without affecting the files that point to them--those files are still available for user manipulation.

3.3.1.3 Shared access to a single copy of files

Sharing files was even more successful than sharing messages. The Military Message Experiment bore out our belief that message handling in a tightly knit organization like CINCPAC is more a matter of managing information than of distributing messages; two thirds of the instructions executed in MME dealt with files or file entries. Shared files played a central role in much of the use of SIGMA; for example, over one fourth of the files opened were *Date Files*. The *Readboard* and the *Action_Log* were special files whose utility was completely associated with sharing the data they provided. During the last 7 months of the experiment there were 315 links established to another user's file (**GET FILE** instructions executed) for a average user population of approximately 70 users, ample evidence that there was considerable sharing among users' files.

3.3.1.4 Central data management

Central data management is what makes sharing messages and files possible. By controlling updates to shared objects through a central process (daemon), we were able to allow users to access objects simultaneously and to do almost anything they wished with them.

3.3.1.5 Foreground-background split

The foreground/background division between the user job and daemons is sound. There are obvious tasks which fall into one or the other category. What is not clear is how to make the division and where to put all the functions. For example, in early versions of SIGMA the user job waited for the daemon to finish his request in case of error. This synchronization was costly and useless. Even at the end of SIGMA development there was talk of putting *Pending* file updates in the user job for on-line users. All in all, the strict separation we enforced in SIGMA simplified our system and gave us flexibility in the placement of functions, both of which were important considerations.

3.3.1.6 Archive

The archive scheme provided by SIGMA is simple and elegant. It was introduced with absolutely no user disruption, since no user involvement was required. Since files remain after messages are archived, the retrieval facilities work as well for archived messages as for on-line messages. The archive function is absolutely required if long term (years) access to information is important. Unless the user can search for required information in the archive without having to bring the archived data back on-line, it will not serve its purpose.

3.3.1.7 Intelligent terminal

By having the terminal contain powerful local editing, most of the user's text processing takes place in the terminal, which provides predictable, timely response. The local editing, multiwindow capabilities, and independent memory management resulted in a natural and highly responsive interaction style. Although no advantage was taken of it, this responsiveness worked equally well over a communication link with long delay (e.g., satellite link or network delay), so SIGMA could comfortably support very remote users with very little difference in performance.

3.3.2 Details about Function and Design

Parts 2 and 4 discuss the way SIGMA looked to the user and how it was built. The following discussion identifies particular lessons we learned that we feel may be of value for others faced with specifying or

designing a message for a military organization similar to CINCPAC. It is difficult to present this information in a uniform and consistent manner; some is quite general, some is very specific, some deals with the function the user sees, some deals with how a feature was implemented. The basic order of presentation follows the organization of Part 2. Where a particular function or design is pertinent, a reference to the discussion of it in an earlier part is supplied.

3.3.2.1 Start-up facilities

SIGMA did not allow users outside the boundaries of the system, i.e., they could not get into the TENEX operating system or modify the system's mode of operation. Users started up the system by simply turning the terminal power on; if the terminal was already turned on but not in use, the user hit !!*CONTROL*!! and !!*RESET*!! keys simultaneously. The terminal then reset itself, the screen cleared, and the message "Terminal Ready" appeared.

There was a very significant delay (20-60 seconds) before SIGMA in fact started. One improved version dramatically cut the delay from 20 seconds to 3 or 4, but the users were not given feedback that SIGMA was starting up; they had to wait until the system got started before anything appeared on the screen. This was unsatisfactory for two reasons: first, the lack of immediate feedback distressed the users; second, SIGMA sometimes did not start, in which case users had to hit the button again. We recommend that any future system make the start-up mechanism extremely simple and automatic. The user should be provided immediate feedback that the start-up (which should be ultrareliable) has begun. Some intelligible error message should appear if start-up fails.

3.3.2.2 The display screen

Three lines of text gave the user general status information: the flash line, the feedback line, and the status line. We observed that this information was essential because it helped the user understand where he was. The working space was broken into a display window and (potentially) a view window. The ability to split the screen and show two separate objects at the same time was very popular--indeed, very important to the success of the system. If a larger screen and more memory were available, we would advocate using the working space more flexibly. There was no particular reason for the working space to be split only into two windows or for text in one of those windows not to be editable; we did it this way only to simplify the control code in the functional module. With a little more complexity (and a larger screen) the window management could be made more flexible, and thus the system made more capable.

Although the facility for splitting the screen and displaying two objects received good grades, a few users indicated that in split-screen mode, the working areas were too small, which has been our opinion for a long time. The computer industry does not offer a large-screen, high-resolution (50 lines of 80 characters) terminal as a standard product, even though such a terminal is quite common in the word processing industry; apparently, the "market" does not demand it. For an application such as automated message handling, the additional cost of such a terminal is justified if a suitable multiwindow capability is provided. An alternative is to provide two separate CRT screens, as provided in NMIC-SS. We feel this alternative is more expensive, less flexible (you cannot view a full page of a single document), and makes too large a package for an office environment. With a higher performance display, you could separate windows with lines across the page. Actually a separator line could easily be a primitive if a terminal were designed from the start to provide multiwindows. The computer research community has demonstrated the use of full-page CRT's with multiple fonts and graphics capabilities in which windows can be set to arbitrary sizes and locations, even overlapping each other. These terminals may be overkill for a system limited to message handling, but if the AMHS eventually evolves into an integrated information/communication system, they will be well worth their added cost.

3.3.2.3 Entering instructions

SIGMA's very sophisticated command language parser not only allows users to enter arguments in any order, but also automatically supplies defaults where appropriate, corrects spelling on mistyped parameters, permits the correction of any typed-in errors before execution, and provides contextual prompting at two levels of detail. The generalized editing facilities available in the terminal applied to the command window, so that the user did not have to operate in several different styles or modes. Users liked the available function keys, although some were obscure and seldom used. On the other hand, some instructions would have worked better as function keys (for example, "find top" and "find bottom"). In general, users had little difficulty with the process of entering instructions, although they had their share of difficulty with the execution of particular instructions (e.g., parameters missing, unexpected results). Very little was said in the evaluation interviews about instruction entry, which we took to be a sign that the large amount of attention we paid to this process paid off.

3.3.2.4 Help system

SIGMA offered two forms of reference material: a reference manual and an on-line help system. Both of these were generated from the same source file; a special program was run against that file to produce both the information for the help database (the on-line system) and text pages for a new copy of the reference manual. There were slight variations (e.g., the reference manual might speak of material on a different page, while the on-line system would provide a direct link to it), but essentially they were identical. Interestingly enough, the users rated the reference manual as "fairly useful" and the help system as "not very useful." We interpret this as simply a function of the communications medium: a CRT is much more difficult to use than a printed manual when one is trying to find some particular information, since with the latter one can flip through the pages, scan large amounts of information quickly, and zero in on the information of interest. The CRT system just did not work very well because it did not fit the way people tend to operate. This was true even though the system had direct imbedded links that made it possible to jump directly from a word of interest to the text describing to that item.

The difficulty also relates to the user's expectations. He simply does not expect a book to automatically present the answers he is looking for; he realizes he has to search for them. In the help database, with the on-line interactive system, users seemed much less patient; their general complaint was that the system never "told" them the information they wanted. Our view is that this remains an unresolved research issue; considerably more intelligence is required in the help system than we were able to put into SIGMA. An on-line help facility is certainly essential, and the implementers of future systems will have to devote considerable attention to content and form. The scheme to force the on-line help to track the reference manual worked well.

3.3.2.5 The tutor system

SIGMA also offered a tutorial system that provided lessons and exercises to help users to learn the system and become more familiar with it. We undertook this with reservations because we had neither the expertise in computer-aided instruction nor the time required to develop high-quality training material.

However CINCPAC's peculiar circumstances (they stand round-the-clock watches) forced us to provide this on-line training capability. In general, the lessons and exercises were reasonably well received; some complained that the lessons were too wordy or badly written and that they presented only a few instructions at a time when users wanted to try several. Had we taken the time to incorporate the users' suggested improvements into the lessons and exercises, we could have improved that part of the service considerably.

Certain aspects of the system could not be contained in exercises, because there was no easy way to simulate interaction with another user (e.g., coordination). A much more sophisticated mechanism is required to solve this problem.

In summary, the users found the tutor useful, but preferred one-on-one training. MITRE's report [26] on the result of training presents this subject in much greater detail.

3.3.2.6 Editing

In SIGMA, editing is done partly inside the terminal and partly through command execution. The naturalness of having editing functions built into the terminal, where they are fingertip responsive and always operational, was very important. The rapid response to local key strokes encouraged the users to feel that they were altering the document itself rather than using an editor (i.e., a third party making the changes for them). The SIGMA editor was not only natural to use but omnipresent--the user did not have to change levels or modes before acquiring the editing capability. We think this is one of SIGMA's particularly strong points.

Early in SIGMA's career, a debate took place about whether the terminals should provide a replace (overstrike) capability, which allows the user to type in new data overstriking the old data at the cursor location. As it was, users replaced by deleting the old and inserting the new as two steps. Since we did not conduct a controlled experiment, we cannot say whether the decision not to provide overstrike was correct (users did not object to its absence). The point to be made here is not that SIGMA should be used as the best model for the features to provide in the local editor (good word processing systems would be better models for that), but that it is important to use a communication protocol to the terminal that isolates the details of its editing from the application program. This way different editing features can be adopted, and new, improved terminals can be incorporated into the system without having to rewrite the application code.

We found that the larger context editor--that is, the part of the editor executed in SIGMA--was generally well accepted, although some potentially useful facilities were never provided (such as a global replace, which allows users to replace every occurrence of a particular string with something else--for instance, every occurrence of "herring" changed to "trout").

We did learn a lesson regarding SIGMA's automatic formatting. At first, SIGMA formatted all text in the display window whenever a command was executed. If a user wanted to move text from a received message into a draft message, SIGMA automatically reformatted using its format algorithm. Since formatting rules are difficult to make flexible enough to suit the unique needs of each user, text was often formatted in a way the users did not want. At the users' insistence, we changed SIGMA so that it would only reformat text when the users hit a particular key; otherwise, we left the information as it was before. Users were allowed to identify a specific area of text to be formatted by placing a mark at the beginning and end of the text. This arrangement was much more successful. We conclude that formatting is a very significant issue; considerable attention to it is needed to provide the right degree of automatic assistance.

3.3.2.7 The structure of the SIGMA messages

In SIGMA, messages are highly structured objects made up of many fields, linked together in a manner that facilitates their manipulation or alteration, as well as the insertion of new information. For this experimental system, we generalized many system features because we wanted the flexibility to respond to users' suggestions quickly and easily. It turned out that for the preparation of messages this was very appropriate. However, incoming messages inherited all the same generality, even though the only alteration made to them was the insertion of a few fields (the Action field, the Information field), the insertion of

comments, and the users' highlighting of specific texts. Although this message structure provided great flexibility, the price paid was that messages were considerably larger (took up more disk space or core space), and more processing was required to link the messages together again for display purposes. Since incoming messages made up 90 percent of the traffic on the system, a severe price in performance and space was paid in exchange for generality which had little utility for this large number of messages. Were we to build a production system, we would look more closely at the presentation of transmitted messages and consider treating them as more simply structured objects, different from preparation messages.

3.3.2.8 Message format

During the design of SIGMA we considered providing facilities to allow users to reformat a message for their personal use, that is, to allow them to change the order in which fields are presented to them. The users advised that this was not a particularly valuable aid. After several years' experience, we saw that they were correct. The only requests we had about the format of messages were a global change in the format of memos and a request to allow users to select for themselves whether a displayed message would start at the top of the message, showing all the addresses, or in the middle of the message, showing the text of the message first. (This does not reorder the fields, but simply changes the initiation point of the display.) The message ordering or template facility of Hermes[14] is a very powerful facility when one extends the message service beyond simple message handling. It certainly deserves investigation for any future message systems, but we cannot make any particular case for its importance as a result of our experience with MME.

3.3.2.9 Message types

SIGMA handles the following types of messages: AUTODIN messages from commands outside of CINCPAC, memos or formal messages internal to CINCPAC, and notes or informal messages between SIGMA users. Although the statistics are not precise, about 90 percent of the messages dealt with in SIGMA were AUTODIN messages. Of the remaining messages, notes were found to be quite useful; memos were used very little, probably because they were used primarily between directorates, and the only directorate with access to SIGMA was J3.

3.3.2.10 Distribution of incoming messages

The LDMX automatically assigned distribution to incoming AUTODIN messages. Those messages that LDMX directed to the J3 directorate were passed on to SIGMA. At first SIGMA simply adopted the assignment made by LDMX for its internal distribution; later, however, it was recognized that the LDMX assignments were not entirely appropriate for SIGMA. For example, LDMX assigned messages to go to J3 himself, when actually his staff assistant, J301, processed them. So a special distribution mapping function was put in SIGMA's Reception daemon to allow the System Control Officer to give alternate directions for incoming messages. This initial implementation permitted messages directed to one user to be redirected to an alternate user or users.

Later, after classified messages were allowed on the system, a potentially serious breach of security occurred when a series of restricted messages were allowed into the SIGMA database. A group of messages that were not to be given broad distribution were passed to SIGMA, which placed them in *Date Files* where anyone with access to SIGMA could see them. A quick fix was installed by having the redistribution function provided in the Reception daemon changed to scan the first six lines of a message for one of several key text

[14] Hermes is a message system produced by Bolt Beranek and Newman Inc

strings set by the System Control Officer. When one of these strings was found, the message was rejected and sent back to the LDMX. As we more carefully examined other possible message reception rules, it became obvious that this quick fix was not sufficient; it was important to provide special scanning on a number of criteria that would override a number of distribution assignments made by the LDMX (necessary for handling Top Secret messages, special handling messages, private messages for a specific officer, etc.). The lesson learned here is that a simple distribution scheme whereby each user provides the criterion for selecting messages he wants to receive is not adequate; overriding principles must apply, which the current LDMX algorithms were not adequate to detect. In fact, the overriding conditions are subtle; they require both significant study and flexible algorithms that can be adjusted in the field. A user's personalizing his own interest profile for internal distribution messages is a very appealing idea, and we have seen it claimed that this is how it should be done in the future. We warn that it is more difficult than it seems to achieve proper distribution.

An even more difficult problem is the assignment of action on messages; again, since only one office can be assigned the action, user interest profiles are inadequate. LDMX has a fairly sophisticated algorithm that sequentially searches a message using up to seven different criteria, and, when it gets its first hit, will assign action on the message on the basis of that criterion. The assignment is often wrong, despite the complexity of the algorithm. Getting the action assignment correct is a nontrivial problem, and user-settable interest profiles do not provide an adequate solution.

Another observation to be made about the distribution experience in SIGMA is the importance of the ROUTE command. **ROUTE** was absolutely essential to J301's task of distributing messages to the J3 staff; it made a difference of at least a factor of two in the time spent handling the distribution of messages. J301 built a set of Selectors which would designate different classes of messages. Applying a particular Selector, he would route with an associated Route list, then he would apply another Selector and associated Route list. He would work his way through the file of incoming messages until he finished the bulk of the processing, then look at the remaining messages individually to distribute them. After a few months of doing this, the J301 people recognized that they were doing a highly automatic process which could be done entirely by SIGMA. J301 asked for a facility which would string together the **RESTRICT/ROUTE** sequences they automatically performed--essentially a form of automated message distribution. Note that this distribution form is not the same as user-settable interest profiles, since it is under central control. Also it is applied after the restricted distribution messages have been culled out.

Another lesson learned regarding the message distribution aspect of SIGMA was that the users liked having the Action and Internal distribution noted on the message itself. They could look at the message and see who was assigned the action (and, in fact, the whole history of how action assignment went); they could also see who had opened the message. The Action assignment was particularly useful as a search criterion on incoming messages; one popular style of reviewing traffic was to look first at all the messages with action assigned to oneself, then at the ones with action to J3 (that is, anyone in J3), then all the rest. The *Action_Log* automatically built by SIGMA was not employed much during the regular use of SIGMA, but during the rerun of the Exercise Power Play it was tested and proved valuable. We suspect the feature should be implemented in a more general fashion, so that special files could be associated with other instructions (e.g., a Draft file for all **CREATE MESSAGE** commands).

We found that a number of users liked to pull messages from the *Date Files* using selectors built beforehand; they could get the messages faster this way, because they did not have to wait for J301's routing. By this means they also discovered messages that LDMX or J301 had distributed improperly, which would never have gotten to them otherwise. LDMX criteria for handling messages were not as dynamic as the users' own selectors. Some users found it very valuable to search the *Date File* database for messages that were not

their action assignment, but that interested them for some other reason. They felt they obtained a much better general overview of CINCPAC activities by perusing the *Date File* in this manner.

One objectionable aspect of interactive message handling was that when a message was forwarded, if the user was not on-line the message just waited until he came on. If the user was expressly interested in a particular message, no human interaction could locate him to tell him a message had arrived. In the manual system, of course, he could leave instructions with his secretary or another officer.

We contemplated providing a "guard" facility as the solution to this. The user could build a special guard list and assign a selector to it; if an incoming message met the selector criterion, the message would be forwarded to the first person on the guard list who was logged in; if no one on the list was on-line, the system operator would be notified. The message would be forwarded to the on-line user along with a highlighted note, saying something like "please call me if this comes in." We never tried the guard feature, but since this lack was cited as an objection to interactive message handling, it might be a good subject for future investigation.

3.3.2.11 Alerts

The Alert mechanism was generally considered a good feature. Users appreciated being able to establish the criteria for causing alerts, for it allowed them to restrict the alerts to their own interests. All users unanimously agreed that this was a nice feature, and they appreciated the fact that personalization took exactly the same form as their standard selector criteria. However, implementation of alerts was somewhat less than ideal. Having a kind of shadow file (called the Alert list) with a maximum of ten entries was not the best way to do it. One problem was that there were only ten entries; if the user did not check them soon enough, some would be lost. More annoying, however, was that even after the alert was processed in the Alert list, the entry still remained in the user's *Pending* file. Often a user had to deal with the entry twice: once in the alert list, and again when he opened his *Pending* file. This was confusing instead of straightforward and automatic. It would have been much more natural if alerts had been implemented as a special aspect of *Pending* files (this would have required keeping the *Pending* file open at all times). When a message meeting the message alert criteria arrived, a user would get the same flash notification. When he pushed the *!!ALERT ON/OFF!!* button, he would get the *Pending* file with the *ALERT_SELECTOR* applied to it. Then taking action on it, he would be acting on the one entry in the *Pending* file. This would have required more mechanism in SIGMA, but the users would have much appreciated having their *Pending* file open at all times even when other files were open as well. In fact, one of the users' main objections was the limitation of having only one object of a type open at one time. This was even more true in the message area, where users said it was very desirable to have several messages open at once.

3.3.2.12 Access to messages

The universally accepted indentifier for an AUTODIN message is the combination of its **From** and **DTG** fields. Unfortunately, this is not guaranteed to be a unique identifier (due to multipart messages, corrected retransmissions, operator errors, etc.). For this reason, SIGMA assigned its own unique message identifier through which users could directly access messages in the database, and no provision was made for directly accessing a message by its From/DTG. To access a message by DTG the user had to open the appropriate *Date File* (thereby closing the current open file) and **RESTRICT** the file with the **From** and **DTG** information. This was usually better than having to search for paper copies in file cabinets, so users gave SIGMA good marks for it.

It would have been more convenient and much faster for the user had we provided direct access by **From/DTG**. This would have required SIGMA to handle the nonuniqueness problem and to parse the **From** and **DTG** fields. Since for many messages in AUTODIN these fields are manually generated (rather than by computer, as for the LDMX), they are prone to be in error (misspelled or not valid); to make some of them would require considerable intelligence in the Reception process and access to the AUTODIN Plain Language Address Tables, or PLAD (the table of legitimate addresses in AUTODIN). Without a table of valid addresses, it would be virtually impossible to recognize misspellings. The table is also needed to provide automatic expansion when the user types the **From** field into the instruction window to access the message. These tables are mentioned in the LDMX, but SIGMA had no access to them. Some special intelligence would also be required to interpret the **DTG**. Again, since users type these in, we found that DTGs were often out of range; it was not unusual to find messages with DTGs for months or even years that had not even occurred yet.

It would have been even nicer to place a link (a pointer to the referenced message) right in an incoming message itself. The user could then have simply pointed at the reference, done a !!*HERE*!!, and pushed the **DISPLAY MESSAGE** key. This would have required SIGMA to parse the reference in the incoming messages and match the **From/DTG** found (as typed in the message) to the **From/DTG** of the messages held in the system. Unfortunately, not all writers of AUTODIN messages use the same rules for indicating a reference message, so parsing the reference is very difficult. The second step would have been to recognize the actual message based on **DTG** and **From**, extracted from the reference in the incoming message. Because users often did not fill in all that information or supplied it in nonstandard form, that linking would be very difficult. For example, when users type in a reference message they often abbreviate the **From** field; a message from "CNO, Washington, D.C." might be referenced as simply "CNO," or a reply to a message might reference "YOUR" (or even "UR") message rather than typing out the full name.

3.3.2.13 Creating outgoing messages

SIGMA provided a variety of ways to create outgoing messages. One was simply to say **CREATE MESSAGE**, which would start the user with a fresh empty message. Another was to **COPY MESSAGE**, which would copy a partially completed message as far as it had already been filled in, but assign it a new message ID date (this allowed the user to have preformatted messages all set up). Another way to generate a message was to reply to a given message.

Readdressal was another avenue of generating outgoing messages, in which a special readdressal message was created. By the time we had the facilities to create outgoing messages, the readdressal form was a fairly trivial extension. It was, however, received by the users with great enthusiasm; some users claimed that it was absolutely the best feature of SIGMA.

We think that this variety of ways of creating messages was very useful. We were surprised that **COPY MESSAGE** was not used much (only 18 times) and ascribe its lack of use to users' ignorance. During the interviews some users asked for a facility to do just that and were surprised when told that it already existed.

One feature of SIGMA outgoing message processing that never quite went smoothly was the automatic filing of the draft message. Whenever a message was created, an entry to that message was automatically made in some file, defaulting to the *Pending* file if another file was not specified. For messages that might have coordination (MEMO and AUTODIN), it was important to ensure that the draft appeared in some file so it could be easily accessed. But for NOTES, which did not allow coordination, it was often confusing. If the user created a message and then sent it immediately, there were two pointers in the file: one to the original preparation message and another to the transmitted message. In SIGMA, these were treated as two separate

entities--necessary because in the coordination process they indeed are two separate entities. If the message is created and immediately released, the preparation version of the message is rather useless, and having two citations often confused the user. We do not have an answer to this. If we did not create that pointer to the preparation message for a note, the message might be lost if the user stopped what he was doing or was interrupted, or if the system crashed.

3.3.2.14 Coordination process

Although in the final MME questionnaire users did state a preference for automated over manual coordination (14 to 2), our conclusion is that the coordination process is very complex and difficult to automate. Users generally found SIGMA's version hard to understand and unnatural to use. Part of the problem is that there is no model of coordination everyone can agree on.

In SIGMA, each user is given his own version as soon as he reads a message sent to him for coordination. This was done so that he could make changes if he wished; he could alter the message and pass it on to somebody else or send it back to the originator; it was even more important if the coordinator wanted to alter the message and immediately release it (a user cannot edit another user's version). When a user adds comments, he is putting the comments on his own version of the message, not the originator's. Somehow, the users never understood this. The originator would look at the message he sent out for coordination and not understand why he didn't see the coordinator's comments on it; he did not realize that he had to look at the coordinator's version to see the coordinator's comments. It would probably have been more understandable to have the user normally get the originator's version and be able only to comment on it. If he wanted to create his own version, he would have to take some other explicit action. One of the keys to the success of coordination in SIGMA is that these various user versions are all tied together as the same message, so that other users automatically have access to each other's versions without having to find them in the database. Another serious problem with on-line automated coordination is the difficulty in presenting editing changes made by a coordinator. That is, if a user edits a message on paper, his remarks are fairly obvious because they appear in red pencil; the parts he didn't want would be crossed out and substitute words added in the margin. This is hard to show on a CRT. There was a plan to do a source compare between two versions and highlight areas that had been changed, but we never had the time to pursue it. This research issue still needs considerable attention before a solution is found.

We once planned that the originator of a message could automate the full coordination scenario (that is, the routing of messages to the various coordinators) without any further intervention. He could set up serial and parallel lists and, when the message was chopped by various coordinators, the system would automatically deliver the message to the next coordinator on the serial list. This proved to be too complex and not what the users wanted; we ended up making the drafter initiate each coordination cycle manually. He would type in the full list of coordinators in a single coordination list and then select any users on the list for the next cycle of coordination; this proved more successful.

Interestingly, the users asked that more status information be provided about each coordinator. We originally started with merely a notation of approval or disapproval. Later, we added at the users' request more information: whether the coordinator edited the message at all, whether the message he had chopped was an old version of the message or the latest, and who did the chopping (that is, the personal name of the individual, not just the role). Even though this made the Chop Status field more complicated and difficult to read, users asked for even more information, such as time of chop.

There were other problems with coordination. One is the same problem of linking to references, discussed earlier in the section on incoming messages. Another serious problem is accessing nonmessage references.

Very often, reference material included books, manuals, letters, or other materials that were not on-line. The coordinator would see the references but would have no way to access them. Another objection the users raised was that there was no way to force the attention of the coordinator. In the manual system, when they handcarried the message around, they knew they were going to get the attention of the coordinator because they stood at his desk. In SIGMA an important message could be sent for coordination, but since there was no way to force the user to respond to it for chopped citation, the originator felt that he had in some sense lost control of the coordination.

A final objection voiced (a very serious one) was that very frequently many of the people on the coordination list did not have access to SIGMA. Coordination could thus be only partly done on SIGMA, and having some of the coordination done off-line and some on-line was not satisfactory since each form only showed what had been done in that medium. Users vastly preferred to stick with one medium all the way.

3.3.2.15 Release of messages

In SIGMA, anyone who received a preparation message, whether he was its originator or the coordinator on the list, was empowered to release it, if he had release authority for that type of message. Usually, everybody had release authority for NOTES, only certain people for MEMOS, and a more restricted set for AUTODIN messages. SIGMA's coordination facilities allowed the releaser to edit the message before he released it; if he did the release, his edited version would be the one that was sent.

When a message was released, SIGMA would create an entirely new message that it passed to AUTODIN. The Preparation message was closed (i.e., it could not be edited any more) and returned to the database. When a *Back_copy* of the message was received from AUTODIN, the message ID of the preparation message was included in the citation produced. Because one could always get back to the preparation message, the history of what people said, who chopped the message, what the comments were, and so forth, was always available. It might have been even better to have provided the link to the preparation message in the SIGMA back copy itself.

We took considerable effort to fix all the difficulties we encountered in releasing outgoing messages to the LDMX. One such problem was making sure that the format of the message as it left SIGMA met AUTODIN requirements, which incidentally we never found completely documented in any single reference. We found, for instance, that if the address field was longer than 54 characters per line or the text field longer than 63 characters per line the message would be rejected; if the user tried to put two addressees on a single line, the message would also be rejected. This was confusing to users who normally did not think about these matters because their secretaries had always written the messages. With an on-line system they tended to write their own messages, and would typically copy a hardcopy form of an incoming message which (for example) showed two addressees on a line (LDMX formatting rules for printing bear no relation to acceptable input forms for messages). In general, our solution was to change the preparation message form to remove some of these problems. We indented the text line so that it was impossible to provide more than the specified number of characters on a line, and we indented the address field even further so that those specifications would not be exceeded. SIGMA did a considerable amount of checking of format and data to catch as many errors as possible before they were given to LDMX; however, SIGMA did not have access to the Plain Language Address table, so it could not validate addresses filled in by the users (it treated them as plain text). It was the job of the LDMX to catch those kinds of errors. If the LDMX found an error that could not be resolved by the Communication Watch Officer, the message would be rejected and sent back to SIGMA. LDMX notified SIGMA of the problem by sending the system operator something called a Service Message, which had no particular relation to the rejected message; there was no automatic way to field that service message and get it to the user who had released the offending message. Instead, the service message went to

the operator of SIGMA, who usually had no idea how to correct the deficiency in the message; in fact, it was very difficult for that operator to manually trace back and find which message had given the problem. Even if he did, by that time the releaser could easily have logged off and gone home.

The basic underlying lesson from all of this is that it is essential to have the automated on-line message system verify a released message and notify the releaser immediately if his message is not suitable for release. If it passes that verification, there should be no subsequent rejection. Essentially, SIGMA should have been able to apply all the rules that LDMX applied; if we had ever learned what they were, we might have been able to do that. As it was, LDMX did its own verification, and we had unhappy users if there was any difficulty.

Another area of contention was the question of who should receive back copies of a released message. Our original proposal was that the releaser, the originator, and all coordinators would get back copies of an outgoing message. This produced some problems in the Communication Center because LDMX automatically made one paper copy for each of those users. This confused the Communication Center personnel if they were users to whom LDMX normally did not distribute. We compromised by sending a back copy only to the releaser. As a result, users often did not know if an outgoing message was in fact out or not; they never got verification from SIGMA because we did not send them back copies, and there was no way for SIGMA to know if LDMX had accepted the message or not. Users had to keep searching the *Date File* to see if the back copy had arrived yet. This is just another example of how important it is to closely integrate the AMHS with the message exchange.

3.3.2.16 File system

SIGMA's facilities for manipulating messages via files consisting of entries that contain pointers to the messages were generally well received. The main objection to the file entry was the poor content of the Subject fields. As mentioned elsewhere, this was not a fault of SIGMA but primarily the result of lack of standards in the AUTODIN community. Because of this, some users said they would have preferred to see the first five lines of text rather than have the system try to pull the subject out. We resisted this suggestion, since it would have made the file entries rather large for the limited size screen. An important attribute of files is that you can see a lot of entries at once; ours had three lines per file entry, so we could only show about six messages on a single screen. A larger size screen would have helped significantly.

Some people feel that users should be able to personalize the format of file entries (that is, to reformat them with different fields of information shown in different places). We never had any request from users to do this; however, it may have proven helpful.

For security purposes SIGMA treated the *Pending* file in a special manner. There were four levels of files, one for each security level; the users were allowed to set them up with combinations of all messages: Top Secret messages in one file and the rest of the messages in a second file, or four separate files. The utility of this was never tested because we never got Top Secret messages; users set their files to have all their messages appear in one file. We suspect that a separate Top Secret *Pending* file would have proven unpopular.

In SIGMA the user could create his own personal files, naming them arbitrarily; this was viewed as a useful feature. Users would like to have been able to control access to files, but it was never high on the priority list because access was limited by other means. Users liked the generality of the file system and its flexibility, although one user complained that he wanted to start a file name with a number rather than with a letter (a feature which made instruction parsing easier and more powerful).

Date Files provided a record of messages that had arrived and a means to access messages by Date Time Group, which turned out to be a very important feature of the system. *Date Files* allowed users to search the database for messages of interest, using a variety of arbitrary criteria to pick out messages.

An important lesson was learned in the generation of *Readboard* files. A significant value of a message handling system like SIGMA lies in the sharing of data. In the manual system *Readboards* are prepared for J3; no one else in the command knows what messages J3's *Readboard* contains. SIGMA's provision of broad access to the *Readboard* prepared for J3 was much appreciated. Division Chiefs scanned the messages in the *Readboard*, discovering what messages J3 had seen and therefore what messages they should be prepared to respond to.

Unfortunately, building the on-line *Readboard* required extra work for the Command Center staff, since they had to build a manual one as well. Because J3 himself actually only read the paper *Readboard*, the command center personnel were not too enthusiastic about building the on-line *Readboard*, but the rest of the user community appreciated the information so much that they insisted it be continued. This pointed up a global lesson already mentioned--that is, the importance of integrating the paper-handling facility with the on-line facility. Had the printer been of adequate quality, the Command Center personnel could build the *Readboard* on-line, then produce the paper copy by printing the file's contents.

SIGMA provided a variety of file manipulation commands such as **FILE**, **MOVE**, **DELETE**, **ACTION**, and so forth. Some of these, like **FILE**, **MOVE**, and **DELETE**, could be applied to multiple entries at one time. But SIGMA did not allow the user to assign action or to forward more than a single entry at once. This sort of inconsistency in the user interface is annoying to users and should have been avoided.

One of the ways of indicating to which messages the command applied was pointing out the message with a !!*HERE*!!. Users requested that they be allowed to provide multiple !!*HERE*!!s and apply the command to all the messages that were **HERE**d; so, for instance, if a user wanted to scan through the message file and **DELETE** some messages, he would just scan, mark each one he did not want with a **HERE**, and execute a **DELETE** command. Again, SIGMA never provided that capability, although it was on the list to be done had there been more time.

We provided a couple of special commands: one was for sorting entries in the file by Date Time Group, another for emptying the contents of a file. These were both found to be useful, especially in building the *Readboard*.

Perhaps the best feature of SIGMA was the facility for selecting subsets of a file. The selector capability and the ability to **RESTRICT** and **AUGMENT** were used a great deal in very effective ways. Allowing users to build their own named selectors was also very valuable and much appreciated. SIGMA offered two ways to build selectors: One was to simply state the criteria in the **CREATE SELECTOR** command itself; the other was to do a series of **RESTRICT** and **AUGMENT** operations until you had the selected subset you wanted, then to save that series of selection steps as a single-named selector. Both techniques were used extensively to build selectors.

It was a nice human interface feature that SIGMA always presented on screen the subset of messages the user was dealing with: if he performed a **RESTRICT** on a file, SIGMA presented him with the message subset that he had restricted, which gave the users a comfortable feeling that they knew what they were dealing with at all times. We discussed with users the possibility of extending the selection criteria to apply to strings from any comments that were hooked on to message entries; we think this would have been a very powerful addition. It could have served as an alternative form of keywording, which moved with the entry to other files.

The keyword feature in SIGMA was not generally well accepted, probably because the keyword was associated with the file and not with the message. This meant that the user could not assign the keyword while displaying the message, the natural time to make such an assignment; he assigned the keyword to the message entry while displaying the file. If the user then moved the message to a different file, the keyword did not go with it. We suspect that a more acceptable implementation would have been to have keyword be one of the message fields that would be extracted as part of a message summary, so users could select on it.

Users requested that they be allowed to do selection on more than one file at once and, in particular, to do selection on a range of dates from the *Date Files.* This would have been very valuable for finding messages when the DTG was not known. As it was the user had to open and search each day's file individually. SIGMA needed some new mechanism to perform such an operation, but it would have been very valuable. We recommend that it be provided in any future message system.

Users expressed a desire to be able to put objects other than messages into files just as in their file drawers they could file any arbitrary objects. SIGMA allowed users to put messages in and to comment on entries, but not to put text objects in the files.

3.3.2.17 Text objects

Users found text objects to be very helpful; they were used for address lists, route lists, general data files, and a variety of applications. People wrote letters as text objects, using the system essentially as a word processing system; they wrote status reports and did many things that had not been envisioned as part of the basic use of the SIGMA system. One of the most important uses of a text object was a general data file where users could put information extracted from messages; since they could move the text of the object in a single, simple way, they had very accurate information with no transcription errors. Some users kept databases in this manner.

Another use of text objects was to generate status reports. Information was extracted from many incoming messages and put into a single text object that formed the basis of a status report that the user could then edit. Or a single outgoing message could be formed from the accumulation of status reports from a number of people. Essentially, this was using the system as a word processing facility with shared data; it proved extremely valuable. One can imagine extending the text object handling into even more word processing applications to enhance the productivity of the organization in areas other than message handling.

3.3.2.18 Sectioned messages

In incoming messages, there were some multipart messages (messages that were broken into two or more separate messages because they were too long for AUTODIN to handle). These were treated as completely separate messages in SIGMA, because they were handed to the system that way. LDMX gave us no indication that such messages were related; although in the body of the text, near the top, one usually could find words something like "Part one of five." The standards for indicating a multipart message were never enunciated or uniformly applied, so it was difficult to use this clue. Multipart messages were especially troublesome when a user tried to readdress one. Each separate part had to be readdressed, all with the same DTG. SIGMA had no capability for assigning the same DTG on two outgoing messages.

3.3.2.19 Access control

One of SIGMA's important features was a command called **GET**, which permitted access to other users' files, selectors, and text objects. Once a user did a **GET** on a file, he could access it as his own and was able to see the changes to the file as they occurred. The **GET** feature allowed--in fact, encouraged--data sharing. There are other ways to provide sharing, such as allowing access directly by name (as was done for viewing the directories of other users); each method has its advantages and disadvantages.

If a system provides access to other users' data, it must also provide facilities to limit that access. Access control was viewed as an important system feature to test; we simply did not know which access control parameters were important, to what degree and on what objects they would be used, the granularity of the access, and so forth. A minimal access control using TENEX to limit access to entire user directories was established early in the program. A fairly complete access-control scheme that would allow users to protect individual objects was proposed, although its very completeness implied a significant effort required to implement it. Other things seemed more important to the users, so this discretionary access-control capability was continually deferred in favor of other features and in the end was never completely implemented. Therefore we do not have the information we would like on the kind of access control that is appropriate.

Since the MME provided terminals only for the J3 directorate, we never were able to address the question of privacy of messages for J3, and their openness outside of J3 itself. There was no access control on the *Date Files*, so anyone with access to SIGMA could get to them. Messages that came to J3 were normally open to all J3 personnel, but if J4 or J5 had had access to SIGMA, it is not clear that freedom of access would have been tolerated; there was some privacy of information, and we suspect that access control problems would have arisen. We believe the appropriate solution is to provide access control to the messages, although possibly separate *Date Files* are required for different directorates. The problem becomes even more severe if the message system is extended to allow access by people outside of the command itself, i.e., outside CINCPAC. At one time it was planned to provide several terminals at CINCPACFLT and PACAF but this was never done, so we never tested that aspect of the system. We believe there are many unanswered questions regarding access control.

3.3.2.20 Archive

Date Files are an index to all messages on the system: not just the messages currently on disk files, but all the messages that have ever been processed. They provide an organized on-line index to the archived messages. Users can even find messages that are one or two years old, searching the *Date Files* with the selector criteria and getting the entries for the massages of interest. All the manipulation can be done with the *Date Files* up to reading the message itself; only then does the user have to break into the archive to retrieve that specific message. Good response from the archive system is essential, howevc If it takes much longer than 10 minutes to retrieve an old message from tape, the users will often prefer to ao without. It would be helpful if the user could indicate the urgency of a retrieval request, so the operators could respond appropriately.

3.3.2.21 Security

A major issue in SIGMA's design was to formulate a security model acceptable to both the users and the people responsible for computer security. The model we designed was based on a security kernel or trusted process that understood a few security primitives and that processed all security-relevant activities. The rest of SIGMA's code could then be written without special verification. The users were asked to make special security acknowledgments and to verify the classification of objects matched with the security lights supplied

on the terminal to insure that SIGMA code did not violate security rules. The security acknowledgments and security lights were used to check on SIGMA's operation, not on the users. The users never understood this; they found the acknowledgments to be somewhat annoying and stupid, but not a major aggravation. We conclude that a future message system based on this user interface to system security would be acceptable. A more thorough treatment of the security issue may be found in [29].

3.3.2.22 User model

SIGMA provided a two-level log-in, since it was important that users be able to log in to the role that they were filling, i.e., J342 or J317, as well as being recorded as the individual who was in fact filling that role. Some roles were always filled by the same individual, others were not. The air desk, for instance, was filled by a number of different people who changed every twelve hours. This two-level log-in was reasonably successful, but it introduced the problem of having two *Pending* files. To get around this, SIGMA provided a concept of *My_ Pending*, which was a way to access one's personal file as distinct from the title file. Unfortunately, when the user was in the *My_Pending* file, the system still thought of him as the title rather than the individual. Certain access control mechanisms did not allow the individual to operate in his *My_ Pending* file as freely as in his *Pending* file. The problem was most serious when the user took some action (e.g., read a message); SIGMA always treated it as the title taking the action, but frequently this violated access-control rights. For instance, the user might make a comment to himself for an entry in his *My_ Pending* file; since the comment was thought to be made by the title rather than the individual, everyone with that title could see the comment, though this was not intended. This area needs more attention before the next message system is built.

3.3.2.23 Printing

MME used a high-speed, low print quality, quiet-operation printer. The machine's special paper did not produce high-contrast print and its glossy coating did not have the feel of normal paper. Almost unanimously, the users objected to the print quality. They wanted to use SIGMA's output for messages to show their boss, to do coordination off-line, and to put the messages in their files; they also would like to have used the printed copies for the word processing types of applications (generating letters, creating briefing memos, etc.). The quality of the printer was simply not good enough for many of these uses.

In future message systems, print quality must be taken into account. The printer itself was successful because of its speed and low noise factor, matters which should also be considered in future choices among printers.

3.3.2.24 News and status

Two bulletin board facilities were provided in SIGMA: SYSTEM NEWS and SYSTEM STATUS. The former was text, printed at log-on, that informed users about the operation of the system and other general information of interest. The latter showed who was logged on and information about them. Both objects could be accessed by special commands. No doubt there was more information that would have been of interest to users (a bulletin board, for example, where users could place messages of general interest for everyone to read, or a calendar of events). Providing such other types of information was never tried on the SIGMA system, but when a message system of this sort has broad distribution, it is very easy to enhance it with tools of this kind to disseminate general information.

3.3.2.25 Miscellaneous

One serious objection to SIGMA's operation was that once a user executed a command, there was no way to abort it. Users strongly objected to this, particularly when response time was poor and the command to be executed took a long time displaying something on the screen. A future message system needs to make it possible for users to inhibit the further execution of an operation if they so wish.

A small point to illustrate a lesson learned. After the first six months or so, we added to the end of messages and text files a special indicator of the end of the object. Previous to that, users were never sure what was the bottom of these objects. This was a small oversight, but a friendly user interface consists of a great number of just these sorts of facilities. No cookbook recipe of all such features exists, so builders of future systems should be responsive to user's requests to add them after the service is in use.

Obviously, good response to the users is essential, but we found that the user in general wants good response for those operations which he views as simple and which in his world should be fast. Probably the best example of this was the displaying of a message. In the paper world, displaying a message means just turning the pages to the message of interest. In SIGMA this could be a very slow process; a user sometimes waited 10 seconds before the message would appear. This is simply unacceptable; it was very important to provide a very fast response time for the simple operation of displaying, editing, scrolling, or paging information. When performing more sophisticated operations, such as **RESTRICT** and **FILE** entries, users are more willing to wait, because they equate this with a time-consuming manual operation that they would otherwise have to perform. However, this too should be relatively swift if the system is to provide an advantage over manual methods rather than simple parity with them.

Users expressed a desire to have a macro facility: the ability to write a sequence of commands and execute it in one step. A number of users became fairly sophisticated at using the message service and would have been able to take great advantage of this macro capability if we had provided it. Doing this cleanly, however, introduces many significant problems, including command syntax, how to handle errors, etc.

When SIGMA allowed only private comments on user files, the facility was not used. When we extended the commenting facility to allow public and personal comments on file entries, it suddenly became very useful; in fact, in the second running of Exercise Power Play, the ability to comment on file entries was used extensively in the *Action_Log*, and for understanding the status of action. SIGMA's commenting facility was another extremely valuable form of data sharing. We did envision a time when users would prefer to be able to see an object without its comments.

An improvement the users suggested was to be able to mark objects with multiple **HERE**s to select parameters of an instruction; we did allow users to mark a single file entry number and execute the **DISPLAY** instruction. However, SIGMA did not allow the users to select multiple entries to execute instructions applying to more than one entry, such as **FILE** and **MOVE**.

Another suggestion by the users was to improve the general algorithm used to clear **HERE**s. If a user executed any !!*HERE*!!s, they would be cleared whenever the user executed the next instruction. However, this prevented a user from doing a **Move** operation on a large document by executing a **FIND TOP**, doing a !!*HERE*!! at the beginning of the document, executing a **FIND BOTTOM**, doing a !!*HERE*!! at the end, then executing a **MOVE TEXT**. The first **HERE** after the **FIND TOP** would have been removed as soon as the **FIND BOTTOM** was executed. This meant the user would have to go to the top of the document, indicate the first !!*HERE*!!, then scroll through the entire document in order to get to the bottom of it to do the second !!*HERE*!! before executing the **MOVE TEXT**. If the document were large, this could take a very long time.

3.3.2.26 What more we could have done

While we still thought SIGMA would be retained at CINCPAC beyond the experiment's end we made a list of improvements we would consider making during the next year. The list is not in any priority order and is intended to show how much farther we might have pushed SIGMA.

Fast **VIEW MESSAGE** -

A message was stored as a collection of separate fields. To display it, each field was accessed separately. It was possible to make a single compound text of the message so that the entire message could be brought up as one uninterpreted string. This would have saved a large amount of processing time. It would only work for **VIEW** since the structure necessary for editing was lost.

Search multiple *Date Files* -

To accomplish this we would have had to expand the **RESTRICT** and **AUGMENT** micros to accept file names, and build new commands out of them. The resulting set of entries would not belong to any file, so certain operations would be excluded (e.g., **COMMENT**, **KEYWORD**). Still, it would have proven useful to those who were searching for a message for which they did not have the **DTG**.

Accept multiple **HERE**s -

We could support multiple **HERE**s of entries in a file to act like entry lists.

Discretionary access control -

We never implemented the full access control scheme that was originally planned. This would have allowed special access to be applied to individual objects by their owners.

Improve reference capabilities -

It was feasible to allow a user to designate a message to be displayed by its **DTG** by one of several schemes. The effort would be significant, but its utility would probably have made it worth doing.

More maintenance support -

SIGMA needed more utility programs to support its CINCPAC system programmers.

Fix VT and terminal deficiencies -

Occasionally, the MME terminal and SIGMA would get into illegal states. SIGMA's solution was to treat it as an error and log the user out. In such a case, SIGMA should reset the terminal and retransmit its entire state. "Centralizing" the VT and, thus, improving its control structure, was a necessary step to making such improvements.

Move to TOPS-20 -

TOPS-20 is an operating system on the PDP-10 which is based on TENEX, but has a number of improvements, especially in the file system. We believe this would make for a more reliable system.

Run terminals at 9600 baud -

The MME terminals are capable of running at 9600 baud. The PDP-11 terminal concentrator needed to be upgraded to support higher speed operation.

Bigger screen terminals -

ISI has modified the HP 2649 basic terminal used for MME to drive a high-resolution large-screen monitor. A small change to the terminal firmware would have allowed operation with these larger screen units without any change to SIGMA. There are ways that the terminal display memory could be expanded above the 12K byte limit, although this has not been tested in the lab.

3.3.2.27 Other user requests

Throughout the experiment users suggested changes to SIGMA which they felt would enhance its utility. A number of features in the later releases of SIGMA were in response to such requests. Many were reasonable, but we did not have time to implement them; these have been described above. The following requests would have been very difficult because they did not fit the structure of SIGMA well. We felt their benefit was marginal compared to the task and therefore did not plan to attempt them. A different system architecture might be able to respond better, so we mention them here.

Full text search - One request we received was the ability to retrieve messages based on a string search of the full text of the message. Because of the size of the message database this would have been an extremely slow operation and would have consumed considerable resources. Furthermore, such a search would only touch those messages which were still on-line. To extend the search to the archive tapes would have been prohibitive. A full test search (say, for key words) of incoming messages as they arrive is not unreasonable (each message gets this treatment just once), but to allow users to trigger such a search on the database of past messages did not seem to us to warrant the effort required (at least for an experiment).

Continuous message display -

Another request we chose to decline was to provide a continuous message display, that is, each message stuck onto the bottom of its predecessor in a file, so the user could simply scroll through them all. This request was in reaction to the slow response of SIGMA to the DISPLAY NEXT (Message) instruction. SIGMA had no concept to support such a structure so it would have required a very large effort to implement. Our view is that such a facility is an artificial way to get around a performance deficiency and that a future message system should address the performance issue directly.

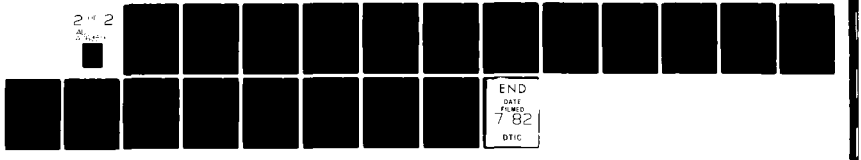
3.4 LESSONS ON SYSTEM DEVELOPMENT AND OPERATION

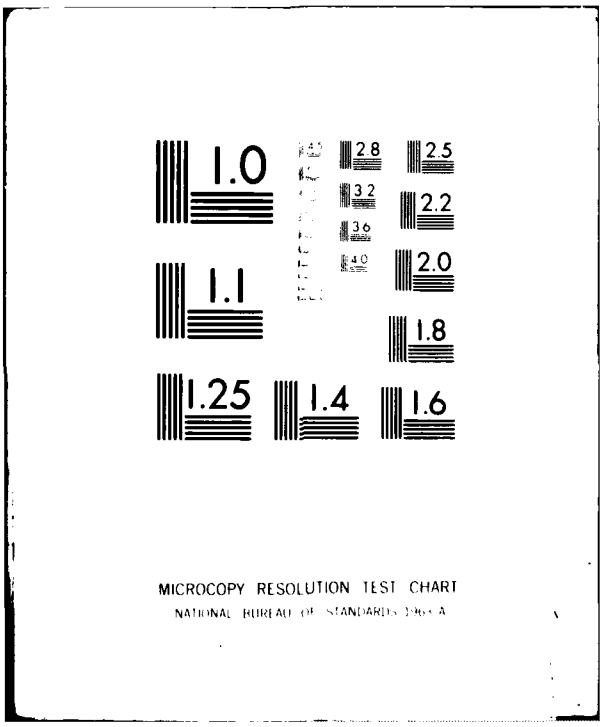
This final section deals with our experience in developing and running a large experimental system of this sort. Much of what we learned is particularly pertinent to experimental systems, but, in general, applies to any large system. In *The SIGMA experience -- A study in the evolutionary design of a large software system* [44], a philosophy and associated principles pertaining to the implementation of a system like SIGMA are presented. In this section issues more specific to the development and operation of SIGMA will be discussed.

3.4.1 An AMHS is a Big System

One often hears about microprocessor-based computers that support message systems; basically they conform to some message transmission protocol and provide a local file system. As noted earlier, we found an AMHS to be a great deal more than that. Besides having a rich set of message-handling functions, it is also a word processor, a database manager and, ultimately, a gateway into a wide range of computer services. When

AD-A116 359 UNIVERSITY OF SOUTHERN CALIFORNIA MARINA DEL REY INFO--ETC F/G 17/2
SIGMA FINAL REPORT. VOLUME V. PART 1-3. INTRODUCTION. FUNCTIONA--ETC(U)
MAY 82 R STOTZ. D WILCZYNSKI. S FINKEL DAHC15-72-C-0308
UNCLASSIFIED ISI/RR-82-94-VOL-5-PT-1 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

coupled with fast access to a large common database, these needs dictate a high level of processing power. And still, this functionality is just a beginning. A consistent and supportive user interface is paramount, adding significant development and processing time to an already large system.

3.4.2 Balance the Goals

Any computer system design begins with a set of goals that can be mapped against the dimension of development time, cost (developmental and operational), function, ease of use, performance, reliability, and flexibility. SIGMA was tightly constrained in nearly all of them with early design decisions made at the expense of performance and reliability. When performance became the critical issue, development time was extended (and operation time of the experiment shortened), and eventually purchase of the KL processor increased the cost.

We learned that all participants must clearly understand what is expected along each dimension and must be careful not to overconstrain the problem. For the SIGMA developers to take the "experimental" nature too seriously was inappropriate when the users did not share the same views.

For a new system that has little precedence, the target specification should be left soft. Had we recognized early in the program that cost was flexible, while the poorly understood performance issue was not, the decision to purchase the KL would have been made much sooner, saving much work and allowing the experiment to start sooner.

3.4.3 Development Environment

We have separated the discussion of the development environment from that of the operating environment. In MME this refers respectively to the computing system at ISI in which we developed SIGMA, and the computing environment at CINCPAC in which we ran SIGMA.

3.4.3.1 Choosing a computer and operating system for development

Our choice to use the TENEX operating system on a PDP-10 processor was not really a choice at all. ISI supports this system fully and (at the time) no other. Since TENEX provides good interactive support for developmental systems, ISI and most other ARPA research sites use it. That SIGMA was developed on TENEX was the natural conclusion. The decision to use the same computer at CINCPAC has similar motivations and will be discussed in section 3.4.4.1.

Part of the reason that TENEX is a popular choice for time-sharing interactive computing is its flexibility, ease of use, and file-sharing capabilities. However, when TENEX is overloaded, these characteristics are lost. SIGMA was developed on a TENEX system being shared with 60 other researchers. Often the only way to get effective work done was to work at night. In comparison, during the two weeks prior to its shipment to CINCPAC, the KL processor system was made available exclusively to the MME project. In that period the Alert facility was coded and revised several times. This was a major piece of of programming that we estimate would have taken at least six weeks under normal conditions, even without the fine tuning in design that we did.

The point is clear. In developing advanced systems with severe time constraints, only the best environments for designers and programmers will do. Psychological and morale problems aside, the best design and implementation will not be discovered by the development team when their development environment blocks their ability to innovate.

3.4.3.2 The programming environment

We decided to implement SIGMA in BLISS, a high-level language oriented toward system programming. This decision was prompted by a number of issues specific to our situation (adequate support, local expertise, efficiency of code on TENEX, etc.). Other than to strongly urge for the use of a high-level programming language, we have little to say about which language should be chosen for any other development.

A much more important issue is the nature of the programming and development environment in which any particular language is embedded. The BLISS environment was totally inadequate and was made bearable only due to our modest effort in enhancing it. We built a package of macros, a modified editor with source line updating, a special batch processor, and a facility that helped in the generation of new SIGMA software releases. Though these few utilities were of the simplest quality, they had a big payoff for us. Any major software development should have the kind of programming environment that is currently beyond the state-of-the-art. Since one does not exist, resources should be dedicated to improving what is available. A quality environment will supply a large set of modular utility packages that can be used by application programmers. As it was, we had to develop our own error-handling package, queue-management routines, interprocess communication protocols, text-handling functions, etc. There were many tools we could have used--a multiprocess runtime debugger, a configuration management system, and so forth, but we did not have time to develop them.

Our strongest advice for the developers of future AMHS is to have a high-quality programming environment. TENEX represents the state-of-the-art for 1970, but it is inappropriate for developing large, highly responsive, reliable applications like AMHS. It is disappointing that even now, five years after the MME project began, there is no widely available programming system obviously superior to the one we used.

3.4.3.3 Developers as users of the system

The earliest version of SIGMA was interfaced to the ARPANET message community. We were able to use it in our everyday work with two important side effects: we exercised the code, and we gained an appreciation of the user's perspective. Unfortunately, SIGMA used so much of the computer's resources that other users on the system complained until we stopped. Of course we realize that the use of the system in the developers' environment might differ significantly from its use in an operational setting, so we don't suggest this as the ultimate solution to system testing. However, many errors appear in boundary conditions that are only unveiled during sustained use. Learning about them all at the operational site is not elegant. It appears to be a distinct advantage to use the system under development in day-to-day work.

There are costly implications in such a strategy: there may be extra work in making the new system compatible with existing systems; a working version of the system ought to be around at all times; the daily work of the developers may be slowed by using the experimental system; and so on. However, the gain ought to be considerable, a better engineered and more reliable system.

3.4.3.4 Developers need access to the operating site system

Security reasons prevented ISI from having remote network access to the CINCPAC system, completely decoupling the development team from the operational system. The combination of on-site system programmers, who were unfamiliar with the SIGMA code, and SIGMA's error package, which produced a snapshot of the runtime state when an unrecoverable error occurred, were our only diagnostic "tools." Error Logs were purged of classified data and mailed back to the development team at ISI every week. Thus, our reaction time to fix bugs was extremely slow and haphazard. Often, the data collected in the Error Logs was

insufficient to identify the problem. To cope with these difficulties, programmers from the development team were sent to Camp Smith to diagnose bugs as they were occurring. Though important, these absences resulted in delays to release schedules.

The nature of an operational context simply makes it unsuitable to debug errors. Users, anxious to get back on-line, are reluctant to call system programmers when errors occur; usually the user simply powered his terminal down for a minute (which normally logged him off) and started up a new job. This, of course, is a direct result of running code that is not fully debugged in an operational environment, an unfortunate consequence of the type of experiment we were conducting and the level of testing we were able to do.

3.4.3.5 Testing

Although we often discussed it, we never expended the large investment required to provide an automated test capability for SIGMA. So prior to each release, we would convene in late night sessions (when we had the machine to ourselves) and run our manual five-user test scenarios. Though the scenarios were constantly changed to reflect new features and the test took over two hours to complete, the procedure was incomplete and did not examine any stress or overload conditions.

It would be too simplistic to just say "we should have provided automated test facilities in SIGMA." Such facilities are not well understood and would have required an immense design and implementation effort to do well. It was not a case of needing test-case generators to test all branches, but one of needing a self-running system, exercising itself to stress interactions between processes and hard-to-anticipate boundary conditions. Such a facility would be expensive to produce. Our conclusion is that while simple automated test tools designed into a system at the outset will have definite payoff, a fully automatic test of large systems is a research topic which deserves more attention.

3.4.3.6 Design for an unreliable environment

We have already stated that a computer system has to be reliable in an operational environment, regardless of its experimental nature. A design corollary is to expect unreliability. The SIGMA design was based on an assumption of reliability in the underlying operating and file system; SIGMA was not built to be robust in the face of operating system errors. A robust design anticipates inconsistencies in the data coming from I/O devices, checks for all anomalous conditions early, and, if an error is detected, takes some sensible action. Unfortunately, one may find a programming effort debilitated by religious attention to these concerns. It is an art and not a science to find the right balance.

To further complicate the problem, error handling during development is different from error handling in the production system. In development you might choose to freeze the state of the machine when an error is detected in order to track down a problem, a solution totally out of place in a production system where you must get the user operating again.

In SIGMA there were many consistency checks throughout the development code; they were compiled out of the production system because of their deleterious impact on performance. We later discovered the need to differentiate between tests on system data and those that tested the running code. Only the latter should have been removed from the production system. Data errors from I/O devices could propagate through the system and cause failures to many users in ways that were extremely difficult to track back to their source.

Debugging is one thing; recovery is another. A robust design also provides operator support for quick and complete recovery when major system crashes do occur. This may entail providing enough redundancy in the

file system to be able to detect smashed data, or utility programs to check the data on disk after a system crash. The completeness of any reconstruction procedure depends, of course, on the effort and expenditure the designers decide upon. For MME, we could not allow a significant loss of any of its major objects, so full and incrematal dumps were taken frequently. Procedures for restoring lost data from those dumps were part of normal TENEX operations. Even so we had one disastrous experience several months after the KL was installed. A disk crash totally destroyed the TENEX files (a weakness of the TENEX file system) and the operator restored the file system incorrectly. When SIGMA came up, many of its files were missing or simply the wrong versions. SIGMA offered no help in detecting the bad state of the file system, other than to crash when someone tried to use inconsistent data. It also provided no assistance to the system programmers who had to incrementally patch the file system back to a consistent state. Although neither the system crash nor the bad restoration of the files was the fault of SIGMA, its passive role extended the impact of the disaster for weeks.

Robustness must also be designed into communication protocols with external devices, notably the terminal. Communication lines are notorious sources of errors, but a good protocol should be able to reduce this error rate to any desired figure (at the price of bandwidth). The protocol between SIGMA and the MME terminal started with a single-character checksum, which allowed one error in 64 to get through. This proved to be too weak, so the protocol was revised to carry a two-character checksum. Yet even this good work had hidden flaws. We discovered later that there was a long resident bug in the PDP-11 terminal concentrator which was never detected because the retransmission protocol entirely masked its effects. One must be careful not to allow the robustness of one part of a system to mask other weak parts.

3.4.4 Operating Environment

The software development environment is important to the developer, and a good one will produce a better product; but of more direct concern to the users is the on-site computer environment.

3.4.4.1 Choice of on-site computer system

Our decision to use a TENEX system at CINCPAC was based primarily on its advantages for development and our limited time and resources. Its general-purpose nature was good for development but turned out to be poor in production. Specifically, file access, terminal I/O, and interprocess communication were all very slow, logically sequential file pages were scattered all over the disk, address space limitation forced process (fork) divisions between which no efficient communication methods existed, and so on. We improved a particular performance problem by rewriting a small terminal output module, but could do little about the other deficiencies.

An operating system tailored to our application would have made a dramatic improvement in performance. Our MME experience also taught us that a computer and message system architecture must support expansion to hundreds of users in a reasonably incremental fashion. It is unlikely that a system will be built to full size from its inception; like the functional performance, we expect the user community to gradually grow. Add to this the importance of reliability and robustness of the underlying computer and operating system, and it seems that the "right" computer system for this application may not be available off the shelf. We suspect the solution lies in the domain of the multiprocessor computer systems designed for ultra-reliable operation, such as Tandem or Delta II. Considering the large software effort it takes to develop an AMHS and the long life expectancy of such a system, any hardware (and software) decisions must be made with a view toward improved and perhaps interchangeable hardware. Building the AMHS for a specific computer with no predesigned plans for migration to alternative hardware simply ignores the reality of hardware progress and immense software costs.

3.4.4.2 Computer operations

There were so many mainline problems in developing SIGMA that we totally underestimated the importance of the operational environment that would house the physical computer and its associated impact on the experiment. At CINCPAC, the MME TENEX system occupied the same room as the WWMCCS system, presumably a stable computer environment. Although no-break power was a specified requirement, the facilities at CINCPAC never supported it. The MME computer ran on ordinary power supplied by Hawaiian Electric through a motor generator and filter equipment installed by the Navy. That liability was compounded by the constant remodeling of the building that housed the computers and most of the terminals and by a power supply that was continually distorted and interrupted. These problems not only disrupted operation of SIGMA, but were a source of stress on the components of the computer. In addition, air conditioning failed on several occasions, components suffered from fungus growth, and the power filter eventually completely failed. The combination of a rather fragile operating system and a hostile environment led to totally unacceptable computer system reliability.

Unfortunately, we did not allocate enough resources to operation and maintenance of the MME environment. For the first 17 months we tried to run the computer with WWMCCS operators, who were insufficiently trained and who had a higher priority commitment to the other machines. That situation improved after the installation of the KL, when a specially trained operations staff was dedicated to MME. By the end of the experiment system uptime began to approach the 98 percent goal.

To underscore the scope covered by what "operations" means, we can point to our problems with the crypto units. The crypto units operated on the communication lines to the two clusters of terminals installed in unclassified areas outside the blockhouse. These units would fall out of synchronization with no indication of failure; to the user it was as though his terminal was disconnected from the computer. To recover, someone first had to diagnose the problem (there was no visual indication) and then call the crypto room to have someone reset the unit. Here was a case of existing hardware, completely out of our control, that greatly influenced a user's view of our system.

3.4.4.3 Understanding goals and maintaining motivation

Besides the now obvious requirements of the physical operating environment, it is important to acquaint the users with the goals for the operational system and to keep them motivated to these goals. Many users view computer systems with skepticism and will have strong biases (some for and some against). Few will fully understand what the presence of a computerized system will mean to them. We found this view particularly true for the MME. For example, several users were reluctant to print messages because they thought the purpose of an AMHS was to eliminate paper. One user, touted as the "most hostile user at CINCPAC," objected to the service because he thought the MME was a field test of a production system that had not had a "cost/benefit analysis" before installation. He continually asked, "What is the problem it is supposed to solve?" Once he understood that the test was to evaluate the utility of a concept which would be part of an eventual cost/benefit analysis, he became very cooperative and gave a fair evaluation of the system.

Even if the goals are understood, having those roles in the user organization that are key to the program filled by qualified, motivated people makes a tremendous difference. When a few important positions were filled by interested officers, acceptance of the whole program shifted dramatically. The experiment proved to be a fragile enterprise that needed competent, active sponsors to see it through.

3.4.5 Conducting an Experiment

Having discussed what we feel we have learned, it is necessary to describe some things we hoped to but did not learn. Because of the limitations of the system (number of users it could support, reliability, time to develop good procedures, integration with the manual system) we were never able to provide any quantitative evaluation of the utility of the system. We were especially anxious to evaluate it in a crisis situation. This would have required supporting users throughout CINCPAC, not just in J3, better computer response time, reliable operations, and a real crisis (or a test of similar proportions). The second running of Exercise Power Play gave encouraging results, but it was not an adequate test.

We would have liked to see the impact of providing informal secure communications between CINCPAC and its subordinate commands. We believe that this ability would have cut down dramatically on the amount of formal traffic and would have led to better information dissemination. This theory remains untested.

Although we think we have contributed to the understanding of the functional design required for an interactive message service for a community like CINCPAC, we do not have much to say about how to build a system that can support 200 or more users, is up 100 percent of the time, and provides all the functionality required with appropriate response.

We did not gain experience in making a multilevel secure system; we did, however, get a sense of the acceptability of a particular model of a user interface to such a system.

The user interface for SIGMA was far advanced for systems of this type. Its universal acceptance redirected our planned efforts in this area to more immediate problems. Thus, many of our goals (user modeling, better user error handling, etc.) went undesigned and untested.

Having built SIGMA, we would like to transfer the developed technology to future AMHS projects. Yet documents can only begin to convey the experience we gained. At one time, we had plans to include programmers from the Naval Command Systems Support Activity on our design team, so they could gain direct experience for future Navy systems. This, however, never took place.

3.4.6 Summary

As is obvious, an AMHS like MME is a large effort. Conflicting goals, highly interactive components, unforseeable developments do much to complicate the project. Many of the problems go beyond cost, project size, and length of development time. If one thing surfaces from our experience, it is that the project as a whole needs to be flexible from the beginning in order to react to changing conditions. A "design, implement, and deliver" paradigm is inappropriate for an AMHS. There are too many sensitive human factors involved that separate an AMHS from other, more easily understood systems. The evolutionary development we are espousing responds not to the limitations of the design team, but to the fundamental nature of the message-handling domain itself.

REFERENCES

1. Abbott, R. J., *A Command Language Processor for Flexible Interface Design*, USC/Information Sciences Institute, ISI/RR-74-24, 1974.

2. Ames, S. R., and W. W. Plummer, *TENEX Security Enhancements*, MITRE Corporation, Technical Report MTR-3217, April 1976.

3. Ames, S. R., and D. R. Oestreicher, "Design of a message processing system for a multilevel secure environment," in *Proceedings of the National Computer Conference*, AFIPS,, 1978. Also appeared as Mitre Corporation Technical Report MTR-3449, June 1978.

4. Bobrow, D. G., J. D. Burchfiel, D. L. Murphy, and R. S. Tomlinson, "TENEX, a paged time sharing system for the PDP-10," *Communications of the ACM* 15, (3), March 1972, 135-143.

5. AUTODIN message CINCPAC 070200Z. August 1975.

6. Military Message Experiment Selection Criteria, 17 September 1976. Prepared by CTEC, Inc., 7777 Leesburg Pike, Falls Church, Virginia 22043.

7. DISTAN Program. Prepared by Naval Electronic Systems Command, Material Acquisition Directorate Telecommunication Division, July 1976.

8. Ellis, T. O., L. Gallenson, J. F. Heafner, and J. T. Melvin, *A Plan for Consolidation and Automation of Military Telecommunications on Oahu*, USC/Information Sciences Institute, ISI/RR-73-12, May 1973.

9. Goodwin, N. C., J. Mitchell, and P. S. Tasker, *Evaluation of ARPANET Message-Handling Systems for Use by the Military*, MITRE Corporation, Technical Report MTR-3096, August 1975.

10. Goodwin, N. C., J. Mitchell, and P. S. Tasker, *Concept of Operations for Message-Handling in CINCPAC*, MITRE Corporation, Technical Report MTR-3323, October 1976.

11. Goodwin, N. C., J. Mitchell, and S. W. Slesinger, *Test Plan for Military Message Handling Experiment*, MITRE Corporation, Technical Report MTR-3268. July 1976.

12. Goodwin, N. C., *Military Message Experiment Baseline Data Report Test Group*, MITRE Corporation, Technical Report MTR-3665, September 1978.

13. Goodwin, N. C., and S. W. Hosmer, *A User-Oriented Evaluation of Computer-Aided Message Handling*, MITRE Co., MTR 3920, April 1980. (MME Final Report, Volume VI, Part 1. [29])

14. Goodwin, N. C., and S. W. Hosmer, *Appendices to a User-Oriented Evaluation of Computer-Aided Message Handling*, MITRE Co., MTR 3946, April 1980. (MME Final Report, Volume VI, Part 2. [29])

15. Heafner, J. F., *A Methodology for Selecting and Refining Man-Computer Languages to Improve Users' Performance*, USC/Information Sciences Institute, ISI/RR-74-21, 1974.

16. Heafner, J. F., *Analysis of Man-Computer Languages: Design and Preliminary Findings*, USC/Information Sciences Institute, ISI/RR-75-34, 1975.

17. Heafner, J. F., and L. H. Miller, *Design Considerations for a Computerized Message Service Based on Tri-Service Operations Personnel at CINCPAC Headquarters, Camp Smith, Oahu*, USC/Information Sciences Institute, Technical Report ISI/WP-3, September 1976.

18. Holg, Chloe, *The Military Message Experiment SIGMA Primer*, USC/Information Sciences Institute, 1977. ISI/TM-77-9.

19. -----, *2645A Display Station Reference Manual*, Hewlett-Packard Company, , 1976.

20. IA Project, Military Message Processing System Design. Unpublished design document. 10 January 1975.

21. Intel, *Intel 8080 Microcomputer System's User's Manual*, Intel Corporation, Technical Report, 1975.

22. Kallander, J. W., N. C. Goodwin, S. Hosmer, C. Smith, D. Fralick, L. Klitzkie, and S. H. Wilson, *Military Message Experiment Mid Experiment Report*, Naval Research Laboratory, NRL Memorandum Report 4094, November 1979.

23. Mandell, R. L., *An Executive Design to Support Military Message Processing Under TENEX*, USC/Information Sciences Institute, ISI/RR-74-25, 1975. draft only

24. Miller, D., *Military Message Handling Experiment Training Requirements*, MITRE Corporation, Technical Report MTR-3263, June 1976.

25. Miller, David G., *Military Message Experiment Training Experience*, MITRE Corporation, Technical Report MTR-3644, August 1978.

26. Miller, D. G., *MME - Final Training Report*, MITRE Corporation, Bedford, Mass., Technical Report MTR-3919, May 1980.

27. -----, Memorandum of Agreement between Director, Defense Advanced Research Projects Agency (DARPA), Commander, Naval Telecommunications Command (NAVTELCOM), Commander, Naval Electronic Systems Command (NAVELEX), and Commander-in-Chief, Pacific (CINCPAC), 1975. Unpublished memorandum.

28. House Appropriations Committee, Report 95-451. U.S. Congress. 21 June 1977.

29. MME Final Report. The *MME Final Report* is being prepared by various individuals and organizations involved in the MME. It will consist of eight volumes; some of the volumes themselves consist of more than one part. References [13] and [14] are Volume VI. For information about how to obtain the other volumes of the *MME Final Report*, contact the Naval Research Laboratory, Washington, D.C. 20375, Attn: Code 7503.

30. *Naval Telecommunications Procedures, Telecommunications Users Manual, NTP3*, 4401 Massachusetts Ave., N.W., Washington, D.C. 20390, 1974.

31. Oestreicher, D., P. Raveling, and R. Stotz, *HP/MME Terminal - Application Specification*, USC/Information Sciences Institute, Technical Report ISI/TM-78-10, March 1978.

32. Rothenberg, J. G., *An Intelligent Tutor: On-line Documentation and Help for a Military Message Service*, USC/Information Sciences Institute, Technical Report ISI/RR-74-26, May 1975.

33. Rothenberg, J. G., *An Editor to Support Military Message Processing Personnel*, USC/Information Sciences Institute, ISI/RR-74-27, June 1975.

34. Rothenberg, J., *DARPA Navy CINCPAC Military Message Experiment SIGMA Message Service Reference Manual*, USC/Information Sciences Institute, Technical Manual 78-11.2, June 1979.

35. Rothenberg, J., "On-line tutorials and documentation for the SIGMA Message Service," in *Proceedings of the National Computer Conference*, AFIPS, June 1979.

36. Slesinger, S. W., and N. C. Goodwin, *Test Procedures for Military Message-Handling Experiment*, MITRE Corporation, Technical Report MTR-3521, October 1977.

37. Oestriecher, D., et al., SIGMA Transition and Deficiency Amelioration Plan, 1977. Unpublished note.

38. Stotz, R., R. Tugender, D. Wilczynski, and D. Oestreicher, "SIGMA -- An interactive message service for the Military Message Experiment," in *Proceedings of the National Computer Conference*, AFIPS,, June 1979.

39. Stotz, R., P. Raveling, and J. Rothenberg, "The terminal for the Military Message Experiment," in *Proceedings of the National Computer Conference*, AFIPS, June 1979.

40. Tangney, J. D., S. R. Ames, and E. L. Burke, *Security Evaluation Criteria for MME Message Service Selection*, MITRE Corporation, Technical Report MTR-3433, June 1977.

41. Tangney, John D., *MME Security Test Procedures*, MITRE Corporation, Technical Report MTR-3615, June 1978.

42. Tugender, R., and D. R. Oestreicher, *Basic Functional Capabilities for a Military Message Processing Service*, USC/Information Sciences Institute, Technical Report ISI/RR-74-23, 1975.

43. Tugender, R., "Maintaining order and consistency in multi-access data," in *Proceedings of the National Computer Conference*, AFIPS,, June 1979.

44. Wilczynski, D., R. Tugender, and D. Oestreicher, "The SIGMA experience -- A study in the evolutionary design of a large software system," in *Proceedings of the National Computer Conference*, AFIPS,, June 1979.

45. Wilczynski, D., R. Stotz, R. Tugender, and R. Lingard, "Message system architecture -- Experience at CINCPAC with the SIGMA System," in *Compcon Spring '80*, IEEE, February 1980.

46. Wilson, S. H., J. W. Kallander, N. M. Thomas III, L. C. Klitzkie, and J. R. Bunch, Jr., *Military Message Experiment Quick Look Report*, Naval Research Laboratory, NRL Memorandum Report 3992, April 1979.

47. Wulf, W. A., D. B. Russell, and A. N. Habermann, "BLISS: A language for systems programming," *Communications of the ACM* 14, (12), December 1971, 780-790.

INDEX


DATE
FILME
7-8